Print Selection

Select All | Clear | Cancel | Print | Print First Page

Section: Page(s): Print Copy: 1

Select?	Document ID	Section(s)	Page(s)	# Pages to print	Database
☑	6748498	all	all	N/A	USPT
☑	6640287	all	all	N/A	USPT
☑	6324654	all	all	N/A	USPT
☑	5890219	all	all	N/A	USPT

Building	Room	Printer
ran	3c18	gbrcptr

Freeform Search

Database: US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Term: L14 and decrement$

Display: 50 **Documents in Display Format:** FRO **Starting with Number** 1

Generate: ○ **Hit List** ◉ **Hit Count** ○ **Side by Side** ○ **Image**

Search Clear Interrupt

Search History

DATE: Saturday, April 02, 2005 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=OR			
L15	L14 and decrement$	10	L15
L14	L13 and track$	17	L14
L13	L12 and increment$	29	L13
L12	L11 and buffer$	45	L12
L11	L10 and register$	70	L11
L10	L8 and ((amount or number) near transfer$)	119	L10
L9	L8 and (amount or number) near (tranfer$)	0	L9
L8	l1 and (data near transfer$)	3336	L8
L7	L5 and decrement$	5	L7
L6	l4 and decrement$	202	L6
L5	L4 and (track$ near unit)	8	L5
L4	L3 and increment$	647	L4
L3	L2 and (register$)	1742	L3
L2	L1 and ((transfer$ or transmit$) near data)	4392	L2
L1	707/$.ccls.	14522	L1

END OF SEARCH HISTORY

1

L15: Entry 6 of 10 File: USPT Mar 30, 1999

7-9

DOCUMENT-IDENTIFIER: US 5890219 A
TITLE: Redundant writing of data to cached storage system

Abstract Text (1):
An integrated cached disk array includes host to global memory (front end) and global memory to disk array (back end) interfaces implemented with dual control processors configured to share substantial resources. Each control processor is responsible for 2 pipelines and respective Direct Multiple Access (DMA) and Direct Single Access (DSA) pipelines, for Global Memory access. Each processor has its own Memory Data Register (MDR) to support DMA/DSA activity. The dual processors each access independent control store RAM, but run the same processor independent control program using an implementation that makes the hardware appear identical from both the X and Y processor sides. Pipelines are extended to add greater depth by incorporating a prefetch mechanism that permits write data to be put out to transceivers awaiting bus access, while two full buffers of assembled memory data are stored in Dual Port Ram and memory data words are assembled in pipeline gate arrays for passing to DPR. Data prefetch mechanisms are included whereby data is made available to the bus going from Global Memory on read operations, prior to the bus being available for an actual data transfer. Two full buffers of read data are transferred from Global Memory and stored in DPR while data words are disassembled in the pipeline gate array, independent of host activity. Timing of system operations is implemented such that there is overlap of backplane requests to transfer data to/from memory, memory selection and data transfer functionality.

Brief Summary Text (4):
A traditional approach to high performance data storage in modern computer systems has been Direct Access Storage Devices ("DASD"), characterized by a single, large, expensive disk drive ("SLED") attached to a host by some standard interface bus. Included in the numerous drawbacks to this approach is a lack of redundancy in the system which means that data is not available when there is a problem with the drive or media. SLED drive data transfer times tend to be orders of magnitude slower than host system data transfer times. The host is moving data at high performance electronic speeds whereas drive data transfer speeds are limited by disk drive physical constraints such as rotational speed, seek time, etc. Thus, typically, DASD has been a primary factor limiting overall information system operating speeds.

Brief Summary Text (5):
Solutions to the relative slowness of SLED have been implemented to take advantage of the fact that very often data required from a drive will be located adjacent to the last data read, a phenomenon referred to as "locality of reference", and to take advantage of the fact that in some cases data required will have been accessed previously in a recent data access, a phenomenon known as "data reuse". Such solutions have involved configuring a fast memory buffer or cache between the host and the drive to yield significant improvements in system performance. When the host requests data, more data than has been requested will be fetched ("prefetching") and held in cache in anticipation that the next read will require adjacent data, in which case it can be retrieved very quickly from cache. Similarly on a write, once the host has transferred data to cache the transfer is complete as far as the host is concerned. The data can then be de-staged from cache to the drive at a convenient time.

Brief Summary Text (6):
Known implementations of cache technology in storage systems, include systems referred to as "Integrated Cached Disk Arrays," ("ICDAs"), which replaced the single large expensive disk drive (SLED) with an array of smaller inexpensive disk drives integrated into a single chassis. The high speed caches implemented in ICDAs yielded improved performance. One family of known ICDA products, known as SYMMETRIX produced by EMC Corporation, Hopkinton, Mass., provides a high reliability array of drives and offers great flexibility in terms of performance

enhancements such as: mirroring; greater data availability; greater data transfer rates over distributed buses; and various levels of redundancy implemented in systems referred to as "RAID systems" ("Redundant Arrays of Inexpensive Disks").

Brief Summary Text (8):
The Global Memory Bus (GMB), between the CA and cache and between the DA and cache in Symmetrix, actually consists of two portions or identical buses designated "A" and "B". The use of two buses improves performance and eliminates a possible single point of failure. Plural sets of Channel Adapters and Disk Adapters, which are generically referred to as "Directors" or "Control Units" (CUs), are assigned to a particular bus based on a physical slot number in a system chassis. The number of available slots is a function of the system type within the family, however, all systems use the dual A and B Global Memory Buses with alternate slots on each bus. Even numbered slots are on the A bus portion of the GMB while odd numbered slots are on the B bus. Each system board on the GMB identifies its position in the chassis by reading a 5-bit slot code encoded on the backplane (referred to as the "SLOT ID"). Each bus has independent arbitration and consists of a 32 bit address bus plus 1 parity, a 64 bit data bus with 8 bits of Error Correction Code (ECC) check bits and a number of control lines with parity. The smallest data transfer that may take place over the GMB is 64 bits during each access (however, byte, word and longword operations are performed within a Director).

Brief Summary Text (10):
The Symmetrix system is designed around a pipelined architecture. A pipeline or pipe in this system is a registered path along which data is clocked to move it from one location to another. Channel Adapters, implemented on a dedicated board that includes two pipelines of its type, make use of pipeline hardware referred to as the "Channel pipe" which moves data between the host Channel and Global Memory. The Channel pipe stages, illustrated in FIG. 1B, include channel receive and transmit FIFOs that implement the Bus and Tag interface. A channel gate array stage functions to assemble/disassemble 64 bit memory words for byte transfers between the Channel interface and a Dual Port Ram (DPR), via Error Detection and Correction circuitry (EDAC is effected by a standard IDT49C465A).

Brief Summary Text (13):
The principle stages of the DMA/DSA pipes are illustrated in FIG. 1D. A Memory Data Register (MDR) provides a 72 bit wide register set, 64 bit data and 8 bits parity, comprised of upper and lower words that can be independently read or written by the 32 bit 68030 microprocessor. The MDR performs the assembly and disassembly of 64 bit Global Memory words (which is performed by the Gate Arrays in the Channel and SCSI pipes). The MDR is also implemented to facilitate byte swapping for data value compatibility between the data values stored in Global Memory by the host and corresponding data processed according to Motorola 68030 byte conventions. The DMA/DSA pipes include the EDAC stage, the Dual Port RAM, and Global Memory.

Brief Summary Text (14):
The single 68030 processor on each Director is used to control the various pipes moving data to/from Global Memory through the pipe(s) on that Director. Other than the DSA pipe, the pipes with multiple memory word transfer capacity function in a substantially similar manner. Each pipe, except for the DSA pipe, is under the control of a 32 bit (write only) command register which is written by the 68030. Pipe commands are routed to and decoded by data transfer control programmable array logic (PAL), which receives command bits and outputs necessary control lines. Pipe status is maintained in a 32 bit (read only) status register which can be read by the 68030.

Brief Summary Text (16):
Dual Port Ram is located on all Directors and is present in all of the data transfer pipes. The DPR serves as a buffer between the Director and the Global Memory. The dual ports to each RAM location facilitate access to each location by different "sides" of the system. One port of the DPR is accessed by a "lower" or "machine side" corresponding to the DPR side accessed by the host, disk array or 68030), and by an "upper" or "global memory side" corresponding to the side accessed by the Global Memory. There is independent arbitration on the "lower" side of the DPR and the "upper" side of the DPR.

Brief Summary Text (17):
Each pipe is allocated a unique block of DPR. Particular locations of DPR are mapped to particular banks/addresses of Global Memory. In using the DPR as a buffer to Global Memory, it

is accessed by a system of pointers. An upper pointer points to the upper side and a lower pointer points to the lower side. There is a unique upper and lower pointer for each pipe on a Director, stored in corresponding upper and lower pointer register files. All pointer values are initially loaded by the 68030 executing a control program and Global Memory writes and reads are effected under control thereof. For transfers to Global Memory, i.e. Global Memory writes, memory words are assembled by the Gate Arrays or MDR and passed to the DPR at the location indicated by the lower pointer. The lower pointer increments until the appropriate sized memory block is written to the DPR (typically 8, sixty four bit memory words). The upper pointer remains unchanged. Transfers from the DPR to the Global Memory commence, typically upon completion of the transfer of the 8 memory words to DPR, whereupon the upper pointer is incremented for each word transfer from DPR to Global Memory. During the transfer to Global Memory, the Gate Array (or MDR depending upon the pipe accessing Global Memory) continues to fill the DPR at the lower/machine side using the lower pointer. For transfers from Global Memory, i.e. Global Memory reads, the processing is effectively the same under control of the 68030 and control program, however in reverse.

Brief Summary Text (18):
Each Global Memory board provides for two port access by the Directors via the two independent buses designated "A" and "B". Global Memory supports a burst transfer feature for all of the pipes whereby 1 to 8 memory words can be transferred sequentially to/from memory for all of the pipes in a single memory access, except for the DSA pipe which only transfers a single memory word at a time. The memory on each board is organized in banks of 8 to facilitate the burst mode transfers. The system operates most efficiently with a burst size of 8 and with a starting address aligned on an 8-word boundary, that is, when the starting address starts at bank 0 and continues up to include bank 7. Each word transfer is clocked by a burst clock signal generated on the memory board, however, the Director is responsible for calculating a final address for each burst transfer. At the end of each transfer, the Director makes a check between address bits and upper pointer bits, which have been incremented by the burst clock, to ensure integrity of the transfer.

Brief Summary Text (20):
The architecture of the Symmetrix ICDA generally facilitates greater data throughput in the data storage system. High reliability and high performance are achieved, among other things, via the dual A and B bus system that allows odd and even memory banks to be accessed with some overlap. Also, performance is enhanced through pipelining including buffering of memory words via the DPR so that memory words can be assembled/disassembled in the gate array pipeline stage and loaded to/from DPR substantially simultaneously with the transfer of memory words to/from Global Memory. However, there are limits on the performance that can be achieved with this ICDA architecture. Specifically, processing speeds are limited by the 33 MHz speed of the 68030 processor. Furthermore, data transfer rates are limited by latency related to servicing pipeline requests. Higher data rates, while difficult to achieve, continue to be desirable for on-line data storage systems.

Brief Summary Text (23):
According to the invention, host to global memory (front end) and global memory to storage array, e.g. disk drive array, (back end) interfaces (generically "Directors"), are each implemented with dual MC68060 control processors each running at 66 MHz, including a first processor referred to as "X" and a second processor referred to as "Y", configured to share substantial resources in order to keep hardware requirements to a minimum. Each control processor is responsible for 2 pipelines designated "A" and "B" and respective Direct Multiple Access (DMA) and Direct Single Access (DSA) pipelines, for Global Memory access. Each processor has its own Memory Data Register (MDR) to support DMA/DSA activity. The dual processors each access independent control store RAM, but run the same processor independent control program using an implementation that makes the hardware appear identical from both the X and Y processor sides.

Brief Summary Text (24):
Pipelines in the integrated cached storage system implementation according to the invention, are extended to add greater depth by incorporating a prefetch mechanism that permits write data to be put out to buffers or transceivers at the backend awaiting bus access, while up to two full 32 word buffers of assembled memory data is stored in Dual Port Ram (DPR), and memory data words are assembled in the pipeline gate arrays for passing to the DPR buffers. Once a first buffer is full it can be sent to memory while a second buffer is being filled, and vice versa,

when the second is full and being emptied the first is filling. The write data is virtually sitting on the bus, which is not enabled for transfer until there is a Director-memory connection, while other data transfer operations are occurring in the pipeline. Similarly, data prefetch mechanisms are included whereby data is made available to the bus going from Global Memory on read operations, prior to the bus being available for an actual data transfer. Two full buffers of read data are transferred from Global Memory and stored in DPR while data words are disassembled in the pipeline gate array, independent of host activity. As data is being taken by the host, one buffer will empty out and is immediately filled while the host continues to take data from the other buffer. This reduces data transfer overhead or actual data transfer time by facilitating memory access and data availability during non-data transfer time. Additionally, timing of system operations is implemented such that there is overlap of backplane requests to transfer data to/from memory, memory selection and data transfer functionality.

Brief Summary Text (25):
Integrated cache according to the invention, i.e. Global Memory, comprises a plurality of memory boards, with variations of 256 Mbyte, 512 Mbyte, or 1 Gbyte of DRAM cache memory. A system can access up to 32 Gbytes of cache. In order to take advantage of the increased processor speeds of the dual control processors according to the invention, the memory boards support burst transfers of up to 32 memory words. The increased burst size, i.e. amount of data transferred between a Dual Port Ram (DPR) and Global Memory, allows for the transfer of 1 to 32 memory words, 64 bits in length, in a single memory access.

Brief Summary Text (26):
The Global Memory is configured such that a memory board having requested data is responsible for driving the clock associated with the transfer of the requested data. During write transfers, the Director controlling the pipeline moving data onto the Global Memory board provides the clock to move data onto the board. Such an implementation, in contrast to some known ICDA systems wherein the memory board or some central memory controller is responsible for all clocks, reduces the turn around time for data transfers to or from memory, i.e. the time from a data request until the data is sent, as clocking is not dependent upon availability of a single, central clocking facility. According to the invention, this clocking scheme increases data set up and hold time for transfers, which increases reliability and data integrity and creates the potential for faster data transfer rates.

Brief Summary Text (31):
One embodiment according to the invention, includes a host to Global Memory interface in which bandwidth to Global Memory is increased by an arbitration mechanism that takes advantage of the dual processor architecture. Access to a Dual Port Ram (DPR) that acts as a buffer to Global Memory is interleaved. Both processors share the same DPR so only one can access it at a time, however, the timing for read and write operations is defined as a function of the availability of the DPR taking into consideration what the other processor is doing.

Brief Summary Text (33):
Features of the invention include a higher speed, higher reliability integrated cached data storage system effecting greater availability of data in the data storage system with dual high speed processors. The architecture is suited for implementation with a large degree of integration via significant amounts of functionality provided in programmable gate array technology. The gate array technology can be programmed in situ by the control processor(s), which leads to enhanced flexibility in updating and/or modifying functionality. The architecture keeps operations involving the DPR and data going to/from Global Memory (i.e. upper side), decoupled from operations involving data going between the DPR and host (i.e. lower side), and in conjunction with a high degree of pipelining avoids significant delays in operations required for data transfers. Unavailability of the backplane bus for data transfers has a limited impact on host storage and retrieval operations. Arbitration of memory bus accesses for all pipes from both processors at the same time so as to effect overlap facilitates significant increases in memory bus bandwidth, i.e. to/from memory, without having to increase clock speeds. A synchronous counter included on each interface, i.e. Director, provides a simple mechanism to improve time correlation between Directors. The 32 bit counter clocked from a one microsecond clock on a selected Director provides a synchronized, high precision time source. The synchronous counter can be used for time stamping of selected system events.

Drawing Description Text (6):
FIG. 1D is a block diagram of a DMA/DSA pipeline for transferring data between Global Memory and a 68030 microprocessor on a Director (CA or DA) of the ICDA of FIG. 1A;

Drawing Description Text (11):
FIGS. 3A-3C are block diagrams of SCSI, Channel and ESCON front end Directors or adapters in a dual processor implementation according to the invention, including SCSI pipes, Channel pipes and ESCON pipes, respectively, for transferring data between a host and Global Memory;

Drawing Description Text (13):
FIG. 3E is a block diagram of a SCSI back end Director according to the invention, for transferring data to a disk array comprising SCSI disk drives;

Drawing Description Text (19):
FIG. 9 is a block diagram illustrating a portion of a pipeline including pipeline stages in a data gate array of FIG. 9 and buffer stages in a Dual Port Ram;

Detailed Description Text (4):
Global Memory in a system according to the invention comprises a plurality of memory boards, four in the present illustrative embodiment, that provide from 256 Mbyte to 4 Gbyte of Dynamic Random Access Memory (DRAM) for cache memory. Each board uses up to 576 16 Mbit DRAM memory devices arranged on both sides of four daughter cards. The global memory boards provide for two port access from one or more controllers or adapters configured either to transfer data between a host and Global Memory (front end adapters), or to transfer data between the Global Memory and the disk array (back end adapters). The Global Memory is accessed by the front end or back end adapters, referred to generically as Directors, via two independent buses, an even bus and an odd bus, and two respective ports designated "A" and "B". The A and B ports each consist of a 32-bit address bus plus 1 parity bit, a 64-bit data bus plus 8 ECC bits, 10 command lines plus 1 parity bit, and bus control lines. The term memory word is used herein to denote a 64 bit plus 8 ECC bit memory word. The memory boards support burst transfers of 64 bit memory words through the A and B ports. The burst transfer feature allows for the sequential transfer of 1 to 32 memory words in a single memory access.

Detailed Description Text (6):
The memory boards have two modes of operation. In a first mode or "operational mode", normal read/write transfers to the Global Memory are effected, as described in detail hereinafter. Operational mode is effectively an on-line/off-line flag signalling the Directors that the Global Memory may be selected for data transfers, including: single memory word write/read via a Directors DSA pipeline; burst read/write of 1 to 32 memory words via an ESCON, Channel or SCSI pipe (depending upon the type of Director). In a second mode or "non-operational mode" a board is non-operational or off-line, and only service cycle reads or writes are possible. Service cycle reads or writes do not write to the Global Memory array and are used for initial set-up and checking memory board status. After power up memory boards wake up in the non-operational state and must be initialized before being available for normal operational memory cycles. During initialization service cycles are used in the non-operational mode to load the segment tables that define what ranges of addresses the board will respond to. Service cycles are also used to read and write status and control information to Maintenance Processors and Memory Controllers on the memory boards.

Detailed Description Text (9):
The main functional components of a memory board 20 are illustrated in FIG. 2B (note that the Maintenance Processor is not shown). Each side, A and B (only one of which is illustrated in FIG. 2B), uses two Data Path ASICs. A High Data Path ASIC 22 handles a portion (bits 71:36) of the 72 bit data bus (consisting of 64 data bits and 8 bits ECC), and a Low Data Path ASIC 24 handles another portion (bits 35:00) of the data bus. The High and Low Data Path ASICS 22, 24 pass data to and receive data from a data register 26 connected to one of the backplane buses (A or B). The High Data Path ASIC 22 and Low Data Path ASIC 24 provide temporary storage and facilitate transfer synchronization for data moving between the system bus or backplane (A or B bus), and odd and even DRAM banks.

Detailed Description Text (10):
A third ASIC, referred to as the Memory Controller ASIC 28, is used for control purposes. The Memory Controller 28 provides clocks and enable signals to the data register 26 and the High

and Low Data Path ASICs 22, 24 to synchronize transfer of data between the system bus and the Data Paths. During write transfers a Director provides the clock to move data into the data register 26 and Data Paths 22, 24. The Memory Controller 28 monitors the clock issued by the Director on a write, to determine transfer completion status.

Detailed Description Text (12):
Global Memory addresses originate at an Address Gate Array Low Memory Address register on a Director receiving a command from a host to perform a read or write operation to the data storage system. The Low Memory Address register is loaded with a 32-bit address to a memory word. As the memory board does not handle data entities smaller than 64-bits, byte addresses are unnecessary. The 32-bit Low Memory address value is output directly on the Global Memory address bus during a memory transfer (i.e. read or write). The 32-bit address, aligned for a 64-bit memory word, provides a capability of up to 32 Gbyte of addressable memory. The address word, along with commands derived from a High Memory Address register on the Director are input to the Memory Controller ASICs via an address and command register 30. In operational mode, the 32-bit address is decoded to provide board selection and DRAM addressing. Board selection is determined by comparing the address to the segment table addresses loaded during initialization. The Memory Controller responds to a valid address by asserting a grant signal (BGRANT).

Detailed Description Text (15):
For integrity purposes, the parity of address information arriving at the daughter card is checked. Buffers on each daughter card generate parity on received addresses. The parity bits so produced are routed back to the Memory Controller where they are compared to parity bits generated internally when the gate array supplied the address to the daughter cards. The address bus arriving at the daughter cards is actually a multiplexed bus which carries row address information during the early part of a cycle and column address information during the latter part. Parity is checked for both row and column addresses.

Detailed Description Text (18):
To effect refresh of the DRAMs, a CAS before RAS refresh mode is used. The refresh interval, (which is specific to the particular DRAMs used as number of refresh cycles required per unit time), is established by the Memory Controller. If neither the A or B ports are active the internal refresh request is asserted at the appropriate interval. If either port is busy, refresh request will not go active until both ports return to the not busy state. However, if the refresh cycle time limit is exceeded before both ports return to the not busy state, refresh will be forced. Refreshing is normally done independently by the two ports, however, in the event of problems with one port, the other port's Memory Controller can facilitate refresh of the entire array at a rate sufficient to maintain data integrity. Refresh cycles are accomplished in a staggered sequential fashion on pairs of banks. Bank 0 and 1 refresh is started, then banks 2 and 3, followed by 4 and 5, and finally banks 6 and 7. An internal counter in the DRAM chips keeps track of the row being refreshed within each bank.

Detailed Description Text (20):
A single Data Path ASIC is illustrated in FIG. 2C. The primary function of the Data Path ASIC is to provide FIFO buffering, on the Global Memory side of the system bus, for read data transfers (i.e. from the memory board to a Director) and write data transfers (i.e. from a Director to the memory board). Data buffering is effected in the Data Path ASICs by means of two dual-port RAMs: a write dual port RAM 32 and a read dual port RAM 34. Use of the dual port RAMs 32, 34 on the memory side of the system bus allows read and write transfers on that side of the bus to be unconstrained by system bus timing, and allows the system bus to be released while the actual transfers to and from the memory array are in progress.

Detailed Description Text (21):
On a write operation, write data from a Director is accompanied by a data clock (CK.sub.-- CU) used on the Global Memory side to load data from the system bus into the data register 26. The bidirectional data bus into the ASIC is enabled for input to the write DPR 32, and data is loaded sequentially into the write DPR 32 strobed by CK.sub.-- CU. Write data read out of the write DPR 32 is passed through an error checker 36, and is then routed via an output select mux 38 alternately to the odd or even outputs as determined by the LSB of the address word. Bi-directional odd and even data buses are enabled for output, and write data is applied to the daughter board DRAMs with synchronization and addressing supplied by the Memory Controller, as described hereinbefore.

Detailed Description Text (22):
On a read operation, read data from the daughter board DRAMs is received by the Data Path via the bidirectional bus enabled for input to the odd and even read DPR 34 inputs. Read data selected via an input select multiplexor 40 is passed through an error checker 42 and loaded into the read DPR 34. A bypass path 44 routes data directly to the output toward the data register 26, bypassing the read DPR when the bypass is enabled by the Memory Controller. If timing and synchronization considerations are such that it is determined that the read DPR 34 will receive/buffer the data, the Data Path will load and read out data to the data register 26. Data emitted from the Data Path is clocked by a Memory Controller clock (CK.sub.-- AMCI on the A side or CK.sub.-- BMCI on the B side). The Memory Controller enables the data register output and provides the accompanying data strobe (CK.sub.-- M) to move data across the system bus.

Detailed Description Text (26):
The lock mechanism according to the invention facilitates a change of memory address within the same memory board during a lock cycle to permit further operations on the locked memory board without issuing other locks. Addressing as controlled by the Memory Controller is implemented such that Low memory address displacement bits 15:00, specifying an address within a 512 Kbyte block, may be changed between accesses to allow addressing within that 512 Kbyte block. However, if low memory address base bits 31:16 change between accesses (which bits are used to specify other memory boards), an error is generated to prevent the locking of more than one board at a time (the error is indicated in a pipe error status register). Such a mechanism allows the system to do house keeping of memory, i.e. cache updating, error handling etc, on the locked memory board under the original lock and prevents any change of address to another memory board while the further operations are ongoing on the locked board. This reduces the number of times the memory board will be locked, unlocked and re-locked to effect operations under lock, effectively reducing the amount of time that other controllers are prevented from accessing that memory board.

Detailed Description Text (29):
As described, Global Memory constituted by individual memory boards configured as described hereinbefore, is written to and read from over the system bus or backplane by controllers referred to generically herein as "Directors", constituted by controllers/adapters transferring data between the Global Memory and the disk array (back end). A plurality of lines or signals comprise the (backend) Director/Global Memory interface. The following signals represent the signals between a Director and a single memory port, i.e. A or B, and are effectively duplicated for a second port.

Detailed Description Text (30):
BD (71:0): 64-bit data (63:0) and 8-bit ECC (71:64) bi-directional bus used for transfer of data between director and global memory.

Detailed Description Text (47):
As described, the basic mechanism for communication between a Director and Global Memory is initiated when a Director selects a memory board by supplying the required parameters and asserting a select line or signal (BSEL.sub.-- IN) for the port to which it is connected. If successfully selected, the memory responds by asserting an associated grant signal (BGRANT). The memory board is thus enabled for data transfer, either read or write. When the required data has been transferred, the memory board returns a 4-bit completion code on status lines (END.sub.-- ST, 3:0), and releases BGRANT. The completion code is sampled by the Director after BGRANT is deasserted at the end of a transfer to determine if the transfer completed successfully or otherwise.

Detailed Description Text (50):
At the front end, one of several types of adapters is configured to transfer data between the host and the Global Memory, as a function of the input/output (I/O) interface of the host. That is, one type of front end adapter has a host interface portion configured to transfer data to/from hosts using an IBM Bus and Tag interface (i.e. the Channel Adapter, CA, described hereinbefore). The technical details of the IBM Bus and Tag interface are set forth in an IBM publication entitled Enterprise Systems Architecture/390, System/360 and System/370 I/O Interface Channel to Control Unit Original Equipment Manufacturer's Information (GA22-6974 OEMI), which is incorporated herein by reference. Another type of front end adapter has a host

interface portion configured to transfer data to/from hosts using an IBM Enterprise Systems Connection interface (i.e. an ESCON Adapter, EA). The technical details of the IBM ESCON interface are described in various IBM publications including INTRODUCING ENTERPRISE SYSTEMS CONNECTION, IBM 3990 ESCON FUNCTION, INSTALLATION AND MIGRATION, and IBM 3990 STORAGE CONTROL ESCON FEATURES PRESENTATION GUIDE, which are incorporated herein by reference. Still another type of front end adapter has a host interface portion configured to transfer data to/from hosts using an "open systems" interface in accordance with the Small Computer Systems Interface standard (i.e. a SCSI Adapter, SA). The technical details of the SCSI interface are described in the SCSI Standard, ANSI/IPCX3.131 199X Rev10D Enhanced Small Computer Systems Interface-2, SCSI-2 which is incorporated herein by reference. Apart from differences in the host interface portions of the several front-end interfaces, which differences are primarily due to accommodation of the particular host I/O interface, the architectures of the front end adapters, i.e. CA, EA and SA, are substantially similar as described hereinafter.

Detailed Description Text (52):
The Directors, implementing respective pipelines for each of the various front end interfaces are illustrated in FIGS. 3A-3C. A front end SCSI Director, for transferring data between a host and Global Memory according to the SCSI standard known in the art, is illustrated in FIG. 3A. The SCSI pipe is the means by which SCSI data is transferred to/from Global Memory. The primary stages of the SCSI pipe include a SCSI chip which implements standard SCSI protocol(s). The interface chip, is a modified SCSI Controller ASIC, part no. 53CFW96-3, manufactured by Symbios Logic. The chip includes selectable SCSI differential 16-bit wide mode capabilities to communicate with a SCSI host (i.e. in the front end application illustrated in FIG. 3A), and is modified in order to additionally provide both the functionality of an NCR 53C94 controller containing drivers to directly drive a single ended 8-bit (narrow) SCSI bus, and to selectably drive a 16-bit single ended (wide) SCSI bus. The same chip is used in the SCSI back end Director to communicate with the disk array (as described hereinafter). The chip implements SCSI standards described in detail in the referenced SCSI standard (which are incorporated herein by reference). The SCSI chip(s) implement the standard bidirectional protocol and pass 16 bit Wide SCSI data in four pipes: an A and B pipe (XA and XB, respectively) on the X side controlled by an X control processor; and an A and B pipe (YA and YB, respectively) on the Y side controlled by a Y control processor. The SCSI data is passed in the four pipes on a bidirectional bus connected to respective SCSI Data Gate Array (GA) devices which assemble/disassemble 64 bit memory words going to/from Global Memory. Error detection and correction (EDAC), and Dual Port Ram (DPR) for buffering, are implemented in the four pipes for data transferred between the SCSI host and Global Memory.

Detailed Description Text (53):
A front end Channel Adapter or Director, for transferring data between a host and Global Memory according to the IBM Bus and Tag (Channel) standard known in the art, is illustrated in FIG. 3B. The principle stages of the Channel pipe include Channel Receive and Transmit FIFOs which receive and transmit Channel data between the host Channel I/O and the Channel Director pipes. Channel data is transmitted in four pipes: an A and B pipe (XA and XB, respectively) on the X side controlled by an X control processor; and an A and B pipe (YA and YB, respectively) on the Y side controlled by a Y control processor. The eight bit Channel data is passed in the four pipes on a bidirectional bus connected to respective Channel Data Gate Array (GA) devices which assemble/disassemble 64 bit memory words going to/from Global Memory. As in other pipes, error detection and correction (EDAC), and Dual Port Ram (DPR) for buffering are implemented in the four pipes for data transferred between the Channel host and Global Memory.

Detailed Description Text (54):
FIG. 3C illustrates an ESCON front end Adapter or Director, for transferring data between a host ESCON I/O and Global Memory according to the IBM Bus ESCON standard known in the art. Data in accordance with the ESCON protocol is received on the ESCON Director by ESCON Receiver/Transmitter Gate Arrays (Rx/Tx GA), which receive and transmit ESCON data between the host ESCON I/O and the ESCON Director pipes in accord with the ESCON I/O protocol. ESCON data is transmitted in two physical pipes that are configured to operate as four pipes, two on each side: an A and B pipe (XA and XB for transmit and receive, respectively) on the X side controlled by an X control processor; and an A and B pipe (YA and YB for transmit and receive, respectively) on the Y side controlled by a Y control processor. Parallel ESCON data is passed in the pipes on a bidirectional bus connected to respective Rx/Tx Gate Array (GA) devices which in turn assemble/disassemble 64 bit memory words. The Rx/Tx GA transmits or receives data to/from an ESCON adapter board that effects data conversion with devices known in the art. The

adapter board (not shown) converts parallel data to serial using a Cypress CY7B923 device (for transmit), or converts serial data to parallel with a Cypress CY7B933 (for receive). As in other pipes, error detection and correction (EDAC), and Dual Port Ram (DPR) for buffering are implemented in the ESCON pipes for data transferred between the ESCON host and Global Memory.

Detailed Description Text (56):
The DMA and DSA pipes, of which there are a total of four - two DMA and two DSA per Director, are substantially similar in topology. An exemplary DMA (or DSA) pipe is illustrated in FIG. 3D. As illustrated, each processor has its own Memory Data Register (MDR) to support DMA/DSA activity. As in the known Symmetrix ICDA, the MDR provides a 72 bit wide register set, 64 bit data and 8 bits parity, comprised of upper and lower words that can be independently read or written by the microprocessor. Similarly, the MDR performs the assembly and disassembly of 64 bit Global Memory words (which is performed by the Gate Arrays in other pipes). The MDR also facilitates byte swapping for data value compatibility between the data values stored in Global Memory by the host and corresponding data processed according to Motorola 68060 byte conventions. Each of the DMA/DSA pipes includes the EDAC stage, the Dual Port RAM, and Global Memory.

Detailed Description Text (57):
At the back-end, i.e. as an interface between the Global Memory and the disk array(s), only one "type" of interface is implemented. A SCSI back end interface is illustrated in FIG. 3E. The back end SCSI Director incorporates the SCSI interface chip which is modified to selectably implement one of several standard SCSI protocols. The interface chip, discussed hereinbefore with respect to the front end SCSI Director illustrated in FIG. 3A, is a modified SCSI Controller ASIC, part no. 53CFW96-3, manufactured by Symbios Logic. The chip is operated in the back end application to selectably drive a 16-bit single ended (wide) SCSI bus (although it can be selectably configured to directly drive a single ended 8-bit (narrow) SCSI bus as well). The SCSI chip(s) implement the standard bidirectional protocol and pass 16 bit Wide SCSI data in four pipes: an A and B pipe (XA and XB, respectively) on the X side controlled by an X control processor; and an A and B pipe (YA and YB, respectively) on the Y side controlled by a Y control processor. At the back end, there is a primary set of two SCSI interfaces and a secondary set of two SCSI interfaces (for a total of four SCSI interface chips) per side. On each side the primary or secondary SCSI interfaces are added for redundancy. The primary and secondary SCSI interfaces are alternately selectable. When two back end interface boards are configured next to each other in a system, the primary of one board can access the drives connected to the secondary of the adjacent board. This redundancy mechanism facilitates data availability even when one back end controller dies. The SCSI data from the selected, i.e. primary or secondary, SCSI interface(s) is passed in the four pipes on a bidirectional bus connected to respective SCSI Data Gate Array (GA) devices which assemble/disassemble 64 bit memory words going to/from Global Memory. Error detection and correction (EDAC), and Dual Port Ram (DPR) for buffering, are implemented in the four pipes for data transferred between the SCSI host and Global Memory. The SCSI back end Director also includes two DMA pipes and two DSA pipes (as illustrated in FIG. 3D), one of each for each of the X and Y processors.

Detailed Description Text (66):
An arbitration mechanism 500 in the ORBIT PLD 314 determines access to all shared resources on the slow bus(es). Access to shared local memory (EPROM, FLASH and NVDRAM) is arbitrated by the ORBIT 314 and granted through its memory module controller 502. Processor access to shared UART and Ethernet communications mechanisms and the TOD are arbitrated by the ORBIT and granted through ORBIT I/O Module controller 504. The ORBIT PLD also maintains shared registers, Read Status Register 506 and Write Control Register 508 accessible to the X and Y processors, which include common registers GENSTAT and GENCON, respectively. The X and Y processors control and monitor the pipeline hardware by means of the GENCON and GENSTAT registers, access to which is determined by the ORBIT PLD.

Detailed Description Text (67):
The arbitration mechanism 500 in the ORBIT PLD 314 enables the X and Y processors to run the same control code, and to access different portions of a shared resource, e.g. NVDRAM or a common register, while appearing to the code to be accessing the same location. Such a "transparency" (or translation) mechanism (FIG. 6, 510), makes the code processor-independent and avoids having to replicate some resources that can otherwise be shared. The transparency mechanism causes the X and Y sides to appear to use the same addresses but in actual fact the addresses are translated by the arbitration transparency mechanism located in the ORBIT PLD.

Detailed Description Text (68):
Eor instance, in running the control program, independent but identical versions, the processors will on occasion need to access NVDRAM 308 which is shared by the X and Y processors on the slow bus. The NVDRAM 308 is used for logging and tracking error information, time stamping via the TOD 312, and also for communication between the X and Y processors 100, 200. The information in the NVDRAM may be critical and thus by the non-volatile nature of the RAM it is retained until manually cleared. As illustrated in FIG. 6, NVDRAM 308 comprises 2 Mbytes of available NVDRAM, divided into two 1 Mbyte sections. According to the ORBIT arbitration mechanism, the first 1 Mbytes is read and writable by a current processor while the second 1 Mbyte is only readable by that processor. The second 1 Mbyte section is read and writable by the other processor, while the first 1 Mbyte section is read only to that other processor. Thus the two processors X and Y have a different overall view, i.e. capabilities, with respect to the same 2 Mbytes of NVDRAM. Each 1 Mbyte section is independently addressable, that is, each section is in reality a different address range. In executing the control program, the processors, X and Y, may need to access some NVDRAM location which to the control program would be the same location. The transparency mechanism causes the X and Y sides to appear to use the same addresses but in actual fact the addresses are translated by the ORBIT arbitration transparency mechanism 510 located in the ORBIT PLD.

Detailed Description Text (70):
The dual 68060 processors 100, 200 run at 50/60/66 MHz from a crystal oscillator/PLL derived clock (depending upon the particular 68060 device used). Referring again to FIG. 4, a 32-bit microprocessor address bus, #MPA 110, 210 is provided for each 68060. The prefix "#" is used throughout the bus nomenclature, wherein "#" refers to either X or Y depending upon the processor side, as the buses are duplicated on each side. A CPU Control device 111, 211, comprised of a Altera EPM7256 Complex Programmable Logic Device (CPLD or PLD), is addressable over the #MPA bus. The CPU Control device 111, 211 carries out much of the 68060 address decoding, generation of control lines to various elements and return of acknowledge back to the processor. It also looks after buffer direction control, initiates DMA/DSA pipe operations, flags the occurrence of bus faults and provides a time-out function for 68060 bus cycles where no transfer acknowledge (TA*) is returned within a specific time period.

Detailed Description Text (71):
The CPU Control CPLD also handles address decode/wait state insertion/acknowledge generation for devices attached to or directly addressable by the 68060, this includes RAM, boot EPROM, Flash, NVDRAM, and I/O space used by pipes, UARTS, interrupt control, TOD, private and shared GENCON/GENSTAT registers which are contained in the ORBIT device 314 and used to control and monitor pipelines. The CPU control CPLD is used to control and monitor pipelines by having the functionality to decode the associated processors address bus and generate read/write, chip select and output enable lines for SCSI primary and secondary SCSI chips, Address gate array, A and B port Data gate arrays, and MDR. Acknowledge in the form of TA* is returned by the CPU Control CPLD to the processor to indicate completion of a bus cycle.

Detailed Description Text (72):
The #MPA bus is buffered to produce #RAMAB 112, 212 and #IOAB 114, 214 buses. The #IOAB 114, 214 bus is further buffered to produce the #IOAB0 116, 216, the #IO1AB 118, 218 and SLOWAB 120 address buses. The SLOWAB bus 120 is shared by the X and Y processors and has "slow" 68060 shared resources on it, i.e. boot EPROM, FLASH, UARTS etc., as discussed hereinbefore.

Detailed Description Text (74):
The control store RAM 102, 202 is attached directly to 32-bit data bus #MPD 130, 230. The data bus is buffered to produce a general data bus, #GDB 132, 232. The buffers between the #MPD and the #GDB can be implemented so as to selectively generate/check byte parity, i.e. for transfers between certain devices. Known 74ABT899 buffers, manufactured by Cypress Semiconductor, are used to generate the #GDB bus 132, 232 from the #MPD and to selectively generate/check byte parity. The 899 buffers have their output enables controlled by the CPU control device 111, 211. The 899s are also responsible for generating/checking parity during processor write/read transfers respectively. Parity is not supported for transfers between all devices. Based on the address present on the processor address bus, the CPU Control device 111, 211 determines if parity should or should not be checked. Where parity should be checked the CPU control CPLD generates the appropriate byte check parity strobes, (XCKPAR*, 3:0), which are routed to the Interrupt Control CPLD 122, 222. Parity errors are latched by the Interrupt Control device 122,

222 and may be monitored via a 68060 register (PGENSTAT2).

Detailed Description Text (75):
The interrupt control gate array 122, 222 is on the #GDB bus. The #GDB bus is further buffered to produce a Memory Data Bus #MDB 134, 234 to which is attached, among other things, the address gate array (ADGA) 124, 224. The ADGA, illustrated in FIG. 7, provides a significant amount of functionality and control related to Dual Port Ram (DPR) 136 used in the pipelines to buffer data transfers to/from Global Memory. Global Memory operations are initiated by the specification of a Low Memory Address, by a 68060, which specifies a starting address for a Global Memory operation. A High Memory address is also specified to provide the limits of the memory transfer. Each of the pipes utilizes the DPR and its pointer system, which is substantially similar to the known pointer system described hereinbefore. The ADGA provides High Memory Address storage 702 for all pipes. It also provides for Low Memory Address storage 704 for each pipe, Upper pointer storage 706 for each pipe, and Lower pointer storage 708 for each pipe. The ADGA is configured with circuitry to monitor upper and lower pointers and generate memory requests issuing Global Memory commands and indicating burst size as required. The ADGA increments memory address and modifies DPR pointers during Global Memory operations. The ADGA maintains pipe status registers 710 and pipe error registers 712 for the A, B, DMA and DSA pipes.

Detailed Description Text (76):
Logic is also implemented in the ADGA to facilitate a limited prefetch operation used by a 68060 control processor in some situations, i.e. DMA pipe memory accesses. A command can be issued by the 68060 to the ADGA to effect prefetch once only to reduce traffic on the backplane. The 68060 command is a non-operational instruction that is used when only a small amount of data, i.e. one to 32 memory words is needed by the control processor. The limited prefetch overrides normal memory operation in that it does not undertake the filling of first and second buffer sections in the DPR. Only the number of words actually specified by the 68060 with issuance of the command will be retrieved and loaded into the DPR to be read by the 68060. The limited prefetch capability has the advantage of reducing bus traffic and system latency as overhead memory operations undertaken by the control processor(s) is reduced by the limited prefetch capability. That is, unnecessary traffic on the backplane, and additional delay of other Directors/memory users is avoided.

Detailed Description Text (78):
If the ADGA receives an indication of a need to use Global Memory, the ADGA will initiate a request to the Upper Machine to access the Global Memory. In the event of multiple requests to use the DPR on the upper side (i.e. the Global Memory side) the ADGA will first arbitrate internally on a per side basis. The ADGA then asserts a request line to the Upper Machine/upper arbiter. Assuming arbitration between the X and Y groups is won, the ADGA will present address and pointer information to the Upper Machine which will proceed to request/select Global Memory and output the upper pointer value on the IB(10:0) lines. From here it is loaded to a counter where it initially passes straight through to become the address to the upper side of DPR. In the case of all transfers except DSA pipe (which only transfers a single word), the counter will increment the upper pointer as each memory word is transferred. The most significant bit is a function of the X or Y group and is generated by the Upper Machine. At the end of a burst transfer, the current value of the incremented pointer, from the counter outputs, is buffered and fed back to the ADGA for error checking.

Detailed Description Text (80):
The ADGA also generates overflow interrupts when a pipe memory address hits a 512k boundary on a Global Memory operation. When overflow is received, the processor has to access ADGA to increment the Low memory address. The ADGA is involved in substantially all reads and writes to Global Memory and is integral in the implementation of a Dual Write feature, as described hereinafter.

Detailed Description Text (82):
The #MDB bus 134, 234 is muxed/buffered by AMD MACH445 devices 144, 244 configured as a bus multiplexor, to generate an #SCSD bus 146, 246. The #SCSD bus is a SCSI data bus to the A and B SCSI chips 126, 128, 226, 228. The SCSI chips in this front end application are configured for 16-bit Wide differential SCSI communication as known in the art. The SCSI chip is capable of sustaining data transfer rates up to 20 Mbytes per second in synchronous mode and 14 Mbytes per second in asynchronous mode. Asynchronous bursts up to 20 Mbytes/second are possible with

proper cabling. The device contains a 32-byte FIFO. All commands, data, status and messages pass through the FIFO on their way to or from the SCSI bus. The FIFO is accessible via the SCSI bus (#SCSD), the microprocessor bus (#MDB), and a DMA bus (#IO 1 AB). Each of the SCSI chips 126, 128, 226, 228 goes to the backplane, i.e. for communication with a SCSI host, through respective SN75LBC976 differential transceivers 148, 150, 248, 250, forming SCSI ports A, B, C and D.

Detailed Description Text (83):
Transfers of data to/from Global Memory effected by the pipes on the SA board involve the DPR 136, which is used by all of the pipes to buffer data transfers effecting pipeline staging and effectively decoupling data transfers from system bus timing. The DPR 136 has an "upper" side whereat transfers between the DPR and the backplane to/from Global Memory are effected, and a "lower" side whereat memory words are assembled/disassembled and transferred to/from the DPR. At the upper side, an FCT16501 bidirectional transceiver provides a final pipeline stage in write operations, basically putting a memory word on the backplane for access to a memory board data path. At the lower side, the DPR is preceded by Error Detection and Correction provided by an IDT 49C465A EDAC implementation (involving sets of IDT EDAC chips known in the art, illustrated as a pair of EDAC sets) 158, 258. The pass through EDAC stage is preceded by memory word assembly/disassembly via the gate array devices 138, 140, 238, 240.

Detailed Description Text (85):
The DPR located on each Director serves as a buffer between the Director and main memory. The dual port features imply that the device has two independent ports which provide access to the same memory locations. The two ports, are referred to in the present illustrative embodiment, as lower or machine side (right) and upper or Global Memory side (left). Although the devices permit apparent simultaneous access from both sides, even to the same location within the device, the nature of the present application is such that this should never occur. In the present illustrative application the control lines for the 8 data lines are connected to the control lines for the parity lines as the devices are effectively used to hold nine data bits. The 72-bits of a memory word (64-bit data and 8 ECC bits) will then fit in 8 physical devices, the least significant 7 devices containing all data while the most significant device contains the most significant data bit and the 8 ECC bits.

Detailed Description Text (86):
DPR is organized as a 4k.times.72-bit (64-bit data+8 ECC bits) buffer using eight 4k.times.9-bit devices. It is divided in two, using the most significant address bit, bit-11. The section from 0 to 7FF is used solely by the pipes under the X processor while the section from 800 to FFF is used solely by the pipes under the Y processor. The buffer is configured such that it is not possible for any of the X processor pipes to use the area of DPR normally used by the Y processor pipes and visa versa. From a programming point of view, to make the code independent of processor, both the X and Y processors appear to operate in the 0 to 7FF range. The hardware however translates addresses in the 0-7FF range to the 800-FFF range for the Y processor. DPR is not directly addressable by the X or Y processors and is in fact addressed by means of pointers.

Detailed Description Text (87):
The DPR is addressed by means of a system of pointers. There is one pointer, the upper pointer for the upper side and another, the lower pointer for the lower side. The pointer value is a memory word rather than a byte address and therefore increments by one as each memory word is transferred. Each pointer originates in a 16-bit register where the least significant 12-bits are actually used as address bits for the DPR. Every pipe on a Director has a unique upper and lower pointer associated with it. These pointer values are stored in registers in the ADGA, as described hereinbefore. As there is only one bus on the lower and one bus on the upper side of DPR, only one pointer can be active on each side at a time. All pointer values are initially loaded by the X or Y processors. The appropriate pointers for the pipes which are active on the upper and lower sides of DPR will be selected by the upper machine/arbiter 252 and a lower machine/arbiter 256, which are responsible for arbitration on the upper and lower sides of the DPR, respectively.

Detailed Description Text (88):
When any pipe, other than the DSA is operating, its pointers, maintained in the ADGA 124, 224 will be incremented automatically as the data transfer progresses. No incrementing of pointers takes place for the DSA pipe which only transfers a single word. For the pointers to be able to

scan through the 64 memory word locations used by each pipe (32 in a first buffer portion and 32 in a second buffer portion, as illustrated and described hereinafter with respect to FIG. 9), the machine has to be able to increment 6-bits of the pointer. Incrementing of the lower pointer is done by the ADGA. Incrementing of the upper pointer is done by an external counter 253, external to the ADGA and at the end of a memory transfer the incremented value is fed back to the ADGA for error checking.

Detailed Description Text (89):
Since the DSA pipe only transfers a single memory word per transfer there is no need to increment the pointer values. In the case of the upper pointer for the DSA pipe, the "next" pointer is still written back to the ADGA by the machine, the difference with the DSA pipe being that the "next" pointer should be the same as the previous pointer as no incrementing takes place. The current pointer value can be read from the ADGA by the controlling X or Y processor at any time. The lower pointer is written/read on bits (15:0) of the data bus while the upper pointer is on bits (31:16). This allows the processor to read or write both pointers in one bus cycle and thus makes for more efficient operation.

Detailed Description Text (90):
Referring now to FIGS. 4 and 9, the DPR is integrally involved in data transfers from Global Memory, i.e. read operations. In a situation where a host requests a read, interface logic (i.e. in the case of the SA board the SCSI ASIC(s), otherwise the ESCON Adapter or Channel CPLD) receives an operation command and provides information to the Director's processor. A 68060 processor reads the information from the interface logic registers and determines what setup is required as a function of the pipe being used. The processor writes the primary memory address, pointers for the DPR and command to the ADGA 124, 224, using the CPU control interface chip 111, 211 to access the ADGA 124, 224. The processor writes the command, transfer size, and direction (i.e. read or write) to the interface logic (SCSI ASIC, ESCON Adapter or Channel CPLD). Once the processor writes the read command to ADGA 124, 224, the ADGA requests the Upper Machine 252 to prefetch two buffers of data (64 memory words). The Upper Machine 252 requests a backplane access to Global Memory and at the same time informs the ADGA 124, 224 to present the memory address, burst size, memory command and DPR pointers (address) for the data to be read from memory. The memory address is sent to a FCT16827 address register 258, and a memory command is sent to FCT806 drivers (not shown), to be sent over the backplane. The DPR pointer (upper) is loaded into counters that are used to generate the address for the DPR.

Detailed Description Text (91):
The DPR 136 is initially empty. The pipeline direction is set to "read from Global Memory" as a result of the memory command, and a pipe is enabled. As the DPR 136 is empty, a request (UREQ*) will be asserted immediately to request the use of global memory. If all is in order, the Director will ultimately succeed in getting the Global Memory bus and selecting the memory board. When the Upper Machine 252 is granted the backplane the memory command, burst and address are sent to Global Memory. The memory in turn is responsible for driving the clock that sends the data to the Director. The memory clock is received on the Director and in turn generates the clocks for the FCT16501 (254, FIG. 4), DPR 136 and the F191 counters 253. A first buffer section (buffer 1, 133, FIG. 9) of the DPR will be filled from global memory, using the upper pointer. A second buffer section (buffer 2, 135, FIG. 4) is still empty so the request (UREQ*) will remain asserted initiating another memory request to fill the second buffer section.

Detailed Description Text (92):
The ADGA 124, 224 maintains the request to the Upper Machine to fill the second buffer. Once the first buffer section 133 has been filled data will be available to move along the pipe, this will be indicated by the ADGA, by asserting a signal (e.g. XA.sub.-- DPR.sub.-- GA.sub.-- RDY*). The pipeline can now remove this data from the low side using the lower pointer while the second buffer section 135 is being filled . The data received by the Director in the DPR full/first buffer section is available to the Assemble/Disassemble GA's (e.g. 138, 140). A prefetch out of the DPR to the GA's of 2 memory words is done to fill two pipeline stages 235, 237 in the GA. This is done by requesting a Lower Machine access. The Lower Machine arbitrates requests from all pipes. When arbitration is granted to the GA, data is sent from the DPR to the GA. The Lower Machine will inform the ADGA which pipe has been selected by a 2 bit code. The ADGA decodes the code lines for the pipe and asserts the pointer (lower) to the DPR. The Lower Machine is responsible for asserting the control logic that will allow the DPR to send the buffered data and the EDAC (IDT49C465) to check this data. A control signal from Lower

Machine output enables the data from the DPR onto the system data bus (SDIO) and another signal latches it into the GA. The ADGA will then internally increment the pointers for the next piece of data. It will also perform some error checking on the transfer to make sure the data was written successfully to the DPR. ADGA provides an Error and Status register that the processor reads for the transfer information.

Detailed Description Text (93):
The GA will continue requesting data from the DPR to fill its pipeline stages by requesting Lower Machine access as long as it has empty pipeline stages. The GA will disassemble the memory words into the format for the appropriate interface logic. The GA will send the data to the interface logic when the Host starts requesting data. The GA and DPR repeat the operation until the transfer count of the Host operation is reached. On the lower side, as the pipe removes data to the GAs and the lower pointer crosses the 20 hex boundary the first buffer section will become available to the high side once again to be re-filled from Global Memory. The second buffer section, if filled, becomes available to the lower side to be emptied.

Detailed Description Text (94):
For burst transfers during reads, the Director calculates the final address for each burst transfer. As a check at the end of each transfer, the Director compares bits 5:0 of the address with bits 5:0 of the upper pointer which has been incremented by the bus clock. The address and pointer bits should be equal, any difference is reported by the director as a read count miss error.

Detailed Description Text (95):
If a DPR buffer empties, the ADGA will automatically request another buffer as described. If the host finishes the data transfer, the interface logic will generate an interrupt. The processor will poll on the interrupt register looking for the interrupt bit to be set. When the processor detects this bit being set it will end the transfer and check for status. The processor will then disable the interface logic and ADGA.

Detailed Description Text (96):
Referring still to FIGS. 4 and 9, in effecting a write operation a host requests a write operation. Interface logic (i.e. SCSI ASIC, ESCON Adapter, Channel CPLD) receives an operation command and provides information to the Director's 68060 processor. The processor reads the information from the interface logic registers, determines what setup is required based on the command specified, and writes the primary memory address, pointers for the DPR 136, and command to the ADGA. The CPU controller 111, 211 is used for these operations to access the ADGA. The processor writes the command, transfer size, and direction (read or write) to the interface logic (SCSI ASIC, ESCON Adapter or Channel CPLD). The write operation starts when the command is written to the interface logic.

Detailed Description Text (97):
The operation starts with the DPR 136 empty . The pipeline direction is set to "write to Global Memory" as a result of the command, and the pipe is enabled. As the DPR is empty, a signal (XA.sub.-- DPR.sub.-- GA.sub.-- RDY*) from the ADGA 124, 224, will be asserted indicating to the pipe data gate array (GA) that the DPR can accept data. When data is received by the Director's Assemble/Disassemble GAs it is assembled in the GA's pipeline stages. When one memory word (8 bytes) is assembled a Lower Machine 256 access is requested by the GA, effectively requesting bus access to the DPR 136. The interface logic allows the Host to continue transferring data, because the built in pipeline stages the GA has (and the pipelining/buffering provided by the GA, EDAC, DPR, and 501 transceiver) effectively decouple data transfer processing from the system bus timing. The Lower Machine arbitrates requests from all pipes. Arbitration is undertaken by the lower machine as described hereinafter, to grant access to the DPR to the GA that has assembled the memory word, and to issue a control signal which enables the GA onto the data bus to the DPR 136. After arbitration is effected according to the suitable arbitration scheme, a 2-bit code is sent to the ADGA on lines corresponding to the pipe that won arbitration. This code identifies one of the four pipes, A, B, DMA or DSA. When a valid code is asserted an appropriate lower pointer is selected, which comes out of the address gate array on the ADL.sub.-- BUS(10:0) lines. These lines are routed to the DPR 136 through a mux 152, the mux outputs being the address lines on the DPR lower side. The mux 152 is used to select between a lower pointer for the X or Y group of pipes under control of the Lower Machine PLD 252. The lower pointer is incremented internally in the ADGA after each memory word is transferred.

Detailed Description Text (99):
As data is presented on the low side the DPR first buffer section 133 starts to fill. The ADGA will then internally increments the pointers for the next piece(s) of data. It will also perform some error checking on the transfer to make sure the data was written successfully to the DPR. ADGA provides an Error and Status register that the processor reads for the transfer status information. The GA will continue sending data to the DPR. When the lower pointer crosses a 20 hex boundary, indicating in the ADGA that the first buffer section 133 of the DPR 136 is full, a memory request will be initiated by the ADGA asserting a request signal (#UREQ). The ADGA switches the pointers to start filling the second buffer with the next memory data. As the second buffer section is still empty, the GA ready signal (XADPR.sub.-- GA.sub.-- RDY#*) remains asserted so the pipe can continue to write data to be assembled in the GA pipeline stages and moved to the second buffer section 135 from the lower side.

Detailed Description Text (100):
When the ADGA detects the first buffer section 1 33 full, the memory request is made to the Upper Machine 252, which requests a backplane access to Global Memory and at the same time informs the ADGA to present the memory address, burst size, memory command and DPR pointers for the first piece of data to be written to memory. The Upper Machine 252 clocks the DPR so data is sitting in the backplane transceivers 254. The memory address is sent to the FCT16827 address register 258 and the memory command is sent to FCT806 drivers. The DPR pointers are loaded into F191 counters that are used to generate the next address for the DPR. However, none of these devices are enabled on the backplane bus. If all is in order and the controller succeeds in getting the global memory bus and selecting the memory board, the data will be transferred from the upper side of the first buffer section 133 of DPR 136 to global memory (while assembly of memory words and filling of the second buffer section 135 are ongoing). When the Upper Machine is granted the backplane, it enables the backplane drivers (254, etc) and sends a backplane clock to memory, to clock in the data. It also sends clocks to the DPR and F191 counters. This operation will be repeated as a function of the size of the burst.

Detailed Description Text (101):
The data being transferred to Global Memory is moved from the DPR 136 tracked by the upper pointer. When the upper pointer crosses a 20 hex boundary the first buffer section 133 will become available to the low side once again. When the transfer is complete the Upper Machine notifies the ADGA which in turn checks for errors. Status is given to the processor in the form of a register.

Detailed Description Text (102):
During the time that the first buffer section is being emptied to Global Memory the pipe continues filling the second buffer section 135 from the low side. When the lower pointer crosses the 40 hex boundary a further memory request will be initiated by the ADGA (by again asserting XUREQ). This time data is transferred from the second buffer section 135 of DPR 136 using the upper pointer. When the upper pointer crosses the 40 hex the second buffer section will become available to the low side once again. At this time the first buffer section 133 should have been emptied and should be available again to the lower side which will start to fill it once again. In effect, the first buffer section and the second buffer section alternately fill from the low side and empty to the upper side (i.e. Global Memory).

Detailed Description Text (103):
If memory is not accessible when the DPR pipe fills the first and second buffer sections of the DPR, the GA ready signal (#ADPR.sub.-- GA.sub.-- RDY*) will de-assert thus blocking any attempted transfers to the DPR by the pipe and effectively stalling pipe operation until data gets moved from DPR to Global memory.

Detailed Description Text (104):
It should be noted that during burst write transfers, the number of 64-bit memory words transferred depends on the burst size and the memory starting address. The Memory Controller on the selected memory board determines the number of memory words it is to receive, from address bits (4:0) and burst bits (4:0) and loads a transfer counter with the appropriate value. The counter decrements as the transfer progresses and produces a termination signal when the transfer is complete. The Director supplies the initial address including the starting bank address on memory address bits BAD(4:0) for a burst transfer involving more than a single word. The Memory Controller alternates addressing to odd and even banks until the required number of

words are transferred, or until a 100 hex byte boundary (32 words) is reached, whichever occurs first, whereupon the memory returns the ending status on END.sub.-- ST(3:0) to the Director. A write count miss error is returned if the actual clock count is not equal to the count calculated on the starting address and burst size determined at the start of the transfer. The status bits are latched by the Upper Machine and provided to the ADGA. The ADGA then reports the status in the error status register.

Detailed Description Text (105):
The integrated cached storage system according to the invention implements a "Dual Write" mode feature or mechanism that facilitates redundancy in data storage by writing the same data to two different locations in Global Memory, while the write data is only processed through the pipeline once. The Dual Write mode is a modification of the write described hereinbefore and is illustrated in FIGS. 17A and 17B. The feature is implemented by setting a bit in the command when issuing a write command to a Director. The ADGA (FIG. 7) includes dual memory address and dual pointer storage for use when the dual write mode is effected. When the mode is set, at the initial part of the operation the processor reads the information from the host interface logic registers, determines what setup is required based on the command specified, and writes the primary memory address, pointers for the DPR 136, a command, and a dual memory address to the ADGA. The dual memory address is the location in Global Memory at which a second copy of the data will be written.

Detailed Description Text (106):
In the dual write mode, the data is only assembled in the DPR one time as described hereinbefore. After the data is written a first time and when the transfer is complete the Upper Machine notifies the ADGA, as usual, which in turn checks for errors. Status is given to the processor in the form of a register. The ADGA then does another Upper Machine request to send the data to the Dual Memory address specified with the command (before any new data can be written into the same DPR buffer). The normal operations for a write are repeated with the memory address being changed to the dual address the processor loaded with the command. The ADGA will reset the DPR pointers to the beginning of the buffer and send the same data to the new address. When the transfer finishes the ADGA then makes the buffer in the DPR available.

Detailed Description Text (107):
Accordingly, in the dual write mode, as illustrated in FIGS. 17A and 17B, one of the data transfer pipelines configured 600 as discussed hereinbefore receives an issued dual write command 602. First and second memory addresses, constituted by the primary and dual memory addresses, are specified and received 604,606 by the ADGA. The write data is assembled/stored 608 in the first stage of the data transfer pipeline and then transferred 610 to one of dthe memory buffers configured in the DPR as described herinbefore. While other memory words are assembled in the first pipeline stage(s) 612 memory words from the buffer memory (DPR) are written to the first/primary address location(s) and to the second/dual memory address location (s) 614. The other memory words subsequently assembled are similarly processed, i.e. transferred from the register pipelinje stage(s) in the assembling GA to the buffer memory/DPR 616 to ultimately be written 618 Global Memory in accordance with either the normal write or dual write ooperations described in detail herinbefore.

Detailed Description Text (108):
The write process, in the normal or dual mode, is continued until the transfer count of the host specified operation is reached. The interface logic will generate an interrupt. The processor will poll on the interrupt register of the interface logic (i.e. SCSI, Channel Adapter etc.) looking for the interrupt bit to be set. When the processor detects this bit as set it will end the transfer and check for status. The processor will then disable the interface logic and ADGA.

Detailed Description Text (109):
Pipeline timing on reads and writes by different processors in the system according to the invention enjoy enhanced processing speeds due to the pipelining described, and the capability to time operations so as to have some overlap between operating processors as well as within a pipelines operation. As illustrated in FIG. 10, when one Director is transferring data (controller0), the bulk of a second (controller 1) Director's overhead can be effected i.e. memory words assembled, DPR buffers filled etc. With the pipeline stages described, upon completion of the overhead by the second Director (controller 1), its data can be effectively sitting on the backplane waiting for the first Director to relinquish the bus upon completion

of the first Directors transfer, while the first director is undertaking a significant portion of its overhead, even during its data transfer. In effect, there will always be data available for transfer across the backplane to/from memory, maximizing utilization and minimizing latency associated with setup time.

Detailed Description Text (113):
In the case of the A and B pipes the request line comes from the data gate array for the respective pipe, while for the DMA and DSA pipes it comes from the ADGA. For DMA/DSA requests, the ADGA receives a signal (#CSDXA) from the CPU Control gate array 111, 211 to indicate the start of a DMA or DSA cycle. From the state of the #IOAB bus bit-9 the CPU Control gate array determines if it is a DMA or DSA cycle and asserts a DMA or DSA request (#CPU.sub.-- REQ.sub.-- DSA or #CPU.sub.-- REQ.sub.-- DMA) as appropriate. The lower machine also receives a unique direction line for each pipe (#A.sub.-- EDC2DPR for the A pipe and #B.sub.-- EDC2DPR for the B pipe). These lines are driven as a function of the pipe command registers in the ADGA. A 68060 line (#SLOWRW*), generated as a function of the 68060 read/write line, determines the direction for the DMA/DSA pipes.

Detailed Description Text (116):
The lower machine PLD receives "direction" signals for each pipe indicating the operation (EDC2DPR for the A and B pipes and SLOWRW* for the DMA/DSA). These signals are used to establish the correct control signal sequence to steer the data along the pipe for read or write operations. The device also captures the identification of the pipe that was active if/when the EDAC detects any single bit errors (SBE) or multi-bit errors (MBE) during a read. The SBE and MBE information is returned to the ADGA where it is reported as error status bits (7:6) of the error status register which is accessible to the 68060.

Detailed Description Text (119):
On the upper side of the DPR 136, transfers between the DPR and Global memory, via the backplane register 254 on the Director and the memory interface, are managed by the Upper machine. Once again there is only one bus on the upper side, however the DPR is partitioned with the first and second buffer sections, of 32 memory words each, assigned for each pipe's use. Each of these sections can independently request global memory. Memory requests are generated when a particular section is full or at the end of a transfer (EOT) in the case of a write operation, or when a section is empty in the case of a read operation. The DPR controlling circuitry, internal to the ADGA monitors the pointer values and generates a global memory request when any DPR section requires service. In the event of multiple simultaneous requests within either of the X or Y group, the ADGA will arbitrate based on a defined priority structure.

Detailed Description Text (121):
Initiation of the memory cycle commences with the upper machine looking for use of the bus/backplane by asserting a signal (REQ.sub.-- OUT*). The signal (REQ.sub.-- OUT) is buffered and routed off the board to a bus master (it should be noted that one of the Directors in a backplane is designated as a bus master as a function of its position in the back plane). Assuming the Director wins arbitration for the backplane the bus master responds by returning the 3-bit slot code which is routed into the Upper Machine PLD on ICU.sub.-- SEL(2:0) lines. The bus master then asserts IRES* into the upper machine to indicate that the code on the ICU.sub.-- SEL(2:0) lines is valid. When the upper machine sees the correct code while IRES* is asserted it then knows it has won the bus/backplane arbitration. The upper machine proceeds by attempting to select the memory board. The ADGA then outputs the pipe's associated upper pointer, memory address and drives the command lines. The upper machine then drives the memory interface and DPR upper side control lines. Data is transferred in a read or write transfer as described hereinbefore. On completion of the memory transfer or in the event of a memory timeout (request or select), the upper machine asserts #UDONE* back to the ADGA. In the case of a DSA write cycle the ADGA then asserts a signal (#DXAAK) back to the CPU Control gate array which then returns TA* to the processor to complete the cycle.

Detailed Description Text (123):
In the illustrative embodiment described herein, only one type of interface is implemented at the back-end (see e.g. FIG. FIG. 3E). The back end adapters moving data between the Global Memory and disk arrays are configured for moving data to/from an array of disks, each of which is configured for data transfers in accordance with a SCSI I/O protocol, which is described in detail in the IEEE technical publication incorporated by reference hereinbefore. The back end

interface is implemented having a dual processor, pipelined architecture that is virtually identical to the architecture of the front end interfaces described hereinbefore.

Current US Cross Reference Classification (1):
707/204

CLAIMS:

1. A method of effecting redundancy in writing at least one memory word to a memory, said method including the steps of:

configuring a data transfer pipeline to transfer said at least one memory word between a data source and said memory, said data transfer pipeline including a first pipeline stage and a second pipeline stage;

issuing a write command to write said at least one memory word;

specifying a first memory address indicating a first location in said memory at which said at least one memory word is to be written and a second memory address indicating a second location in said memory at which said at least one memory word is to be written;

assembling said at least one memory word in said first pipeline stage;

transferring said at least one memory word from said first pipeline stage to said second pipeline stage;

assembling a second at least one memory word in said first pipeline stage substantially concurrently with said step of transferring said at least one memory word from said first pipeline stage to said second pipeline stage;

writing said at least one memory word to said first memory address and writing said at least one memory word to said second memory address; and

transferring said second at least one memory word from said first pipeline stage to said second pipeline stage substantially concurrently with said step of writing said at least one memory word to said first memory address and writing said at least one memory word to said second memory address.

2. The method of claim 1 wherein said first pipeline stage comprises a register receiving data from said data source and is configured to assemble said data into said at least one memory word.

3. The method of claim 1 wherein said first pipeline stage is configured having a first register to assemble and hold one memory word at a time and a second register receiving an assembled memory word from said first register, and said second pipeline stage comprises a dual port ram (DPR) having a plurality of locations and is configured to receive from said first pipeline stage and store a plurality of memory words.

7. A method of pipelining data from a data source for redundant writing to a data destination, said method comprising the steps of:

receiving data from said data source during a write operation and assembling said data into a plurality of data destination words;

receiving a first address indicating a first location in said data destination at which said plurality of data destination words are to be written and a second address indicating a second location in said data destination at which said plurality of data destination words are to be written;

storing each of said plurality of data destination words individually in at least one register pipeline stage;

configuring a first and a second buffer memory, said first and said second buffer memory being

in communication with said at least one register pipeline stage to receive respective ones of each of said plurality of data destination words therefrom;

transferring a first write portion of said plurality of data destination words into said first buffer memory until a first selected number of data destination words are transferred to said first buffer memory;

transferring said first write portion of said plurality of data destination words from said first buffer memory to said first location in said data destination until said first selected number of data destination words are transferred from said first buffer memory to said data destination;

transferring a second write portion of said plurality of data destination words assembled and stored in said at least one register pipeline stage into said second buffer memory until a second selected number of data destination words are transferred to said second buffer memory, while said step of transferring said first write portion of said plurality of data destination words from said first buffer memory to said data destination is taking place;

transferring said first write portion of said plurality of data destination words from said first buffer memory to said second location in said data destination until said first selected number of data destination words are transferred from said first buffer memory to said data destination; and

transferring said second write portion of said plurality of data destination words from said second buffer memory to said data destination until said second selected number of data destination words are transferred from said second buffer memory to said data destination, while said first buffer memory is available to receive another write portion of said plurality of data destination words assembled for transfer by said at least one register pipeline stage.

8. The method of claim 7 wherein said at least one register pipeline stage comprises a first register pipeline stage and a second register pipeline stage, said first register pipeline stage configured to receive said data and assemble said data into respective ones of said plurality of data destination words, and said second register pipeline stage is configured to receive said respective ones of said plurality of data destination words from said first register pipeline stage and to transfer said respective ones of said plurality of data destination words to one of said first buffer memory and said second buffer memory.

9. The method of claim 7 wherein said first and second buffer memory are configured using dual port random access memory (DPR) and wherein a first port of said DPR is configured for handling data transferred between said DPR and said data source, and a second port is configured for handling data transferred between said DPR and said data destination.

10. The method of claim 7 wherein each of said plurality of data destination words is processed by an error detection and correction mechanism prior to being received by said first buffer memory and said second buffer memory.

11. An apparatus effecting redundancy in writing data from a data source to a memory as at least one memory word transferred over a memory bus, comprising:

a data transfer pipeline configured to transfer said at least one memory word between said data source and said memory, said data transfer pipeline including at least a first pipeline stage and a second pipeline stage, said first pipeline stage comprising at least one register receiving said data from said data source and assembling said at least one memory word from said data, said second pipeline stage comprising at least one memory buffer receiving said at least one memory word from said first pipeline stage;

memory address registers storing a first memory address indicating a first location in said memory at which said at least one memory word is to be written and storing a second location in said memory at which said at least one memory word is to be written; and

a memory bus transmitting device receiving said at least one memory word from said at least one memory buffer and putting said at least one memory word on said memory bus a first time to write said memory word to said first location and putting said at least one memory word on said

memory bus a second time to write to said second location.

12. The apparatus of claim 11 wherein said first pipeline stage comprising at least one register receiving said data from said data source and assembling said at least one memory word from said data, includes a first register receiving said data as at least one data element from said data source, a second register receiving a plurality of said at least one data element from said first register and assembling a memory word of said plurality of said at least one data element therein, and a third register receiving said memory word from said second register for transferring to said second pipeline stage.

13. The apparatus of claim 11 wherein said second pipeline stage comprising at least one memory buffer receiving said at least one memory word from said first pipeline stage includes a first memory buffer and a second memory buffer and wherein one of said first memory buffer and said second memory buffer receives memory words from said first pipeline stage while the other of said first memory buffer and said second memory buffer provides memory to said memory bus transmitting device for writing to said memory.

Previous Doc Next Doc Go to Doc#

to ensure integrity of the transfer

DOCUMENT-IDENTIFIER: US 6748498 B2

TITLE: Scalable multiprocessor system and cache coherence method implementing store-conditional memory transactions while an associated directory entry is encoded as a coarse bit vector

Abstract Text (1):
A system including a plurality of processor nodes is configured to execute a cache coherence protocol that avoids the use of negative acknowledgments and ordering requirements on the underlying transaction-message interconnect/network, and implements store-conditional memory transactions. A store-conditional memory transaction succeeds if a directory tracking the state of a memory line of information unambiguously indicates that the requesting node is the exclusive owner of the memory line, if the directory ambiguously indicates that the requesting node is sharing the memory line and the requesting node is in fact sharing the memory line, or if the directory unambiguously indicates that the requesting node is sharing the memory line. The store-conditional memory transaction fails if the directory unambiguously indicates that the requesting node is not sharing the memory line, or if the directory ambiguously indicates that the requesting node may be sharing the memory line and the requesting node is in fact not sharing the memory line.

Brief Summary Text (17):
In one aspect of the invention, a store-conditional memory transaction succeeds if a directory tracking, the state of a memory line of information unambiguously indicates that the requesting node is the exclusive owner of the memory line of information, if the directory ambiguously indicates that the requesting node is sharing the memory line of information and the requesting node is in fact sharing the memory line of information, or if the directory unambiguously indicates that the requesting node is sharing the memory line of information. But the store-conditional memory transaction fails if the directory unambiguously indicates that the requesting node is not sharing the memory line of information.

Drawing Description Text (6):
FIG. 4 depicts a directory data structure for keeping track of which nodes of the system have copies of each line of memory data.

Drawing Description Text (10):
FIG. 6C depicts a subset of the fields of each TSRF entry in the Transient State Register File (TSRF) of the protocol engine of FIG. 5.

Drawing Description Text (13):
FIG. 7C depicts a state transition diagram for any single one of the TSRF entries in the Transient State Register File (TSRF) of the protocol engine of FIG. 5.

Detailed Description Text (2):
All specific quantities (such as numbers of processors, number of nodes, memory sizes, bit sizes of data structures, operating speeds of components, number of interfaces, number of memory locations in buffers, numbers of cache lines), as well as the sizes and number of components in various data structures, disclosed in this document, are provided solely for purposes of explaining the operation of one particular embodiment. These quantities will typically vary, sometimes significantly, from one implementation of the invention to another.

Detailed Description Text (12):
TSRF: Transient State Register File.

Detailed Description Text (17):
From the point of view of a programmer, the PC 106 on the I/O node 104 is indistinguishable

from a PC 106 included on the processor node 102. Similarly, memory at the I/O node 104 fully participates in the global cache coherence scheme of the multiprocessor system 100 (FIG. 1). The presence of a PC 106 on the I/O node 104 provides several benefits. For instance, it enables optimizations such as scheduling device drivers on this processor for lower latency access to I/O, or virtualization of the interface to various I/O devices (e.g., by having the PC 106 interpret accesses to virtual control registers). Except for the PCI/X interface 142, most of the modules on the I/O node 104 are identical in design to those on the processor node 102. For example, the same first-level data cache module (dL1) 110 that is used with the PCs 106 is also used to interface to the PCI/X module 142. The dL1 module 110 also provides the PCI/X interface 142 with address translation, access to I/O space registers, and interrupt generation. The I/O node 104 may also be customized to support other I/O standards such as Fiber Channel and System I/O.

Detailed Description Text (22):
In a preferred embodiment, the PC 106 uses a single-issue, in-order design capable of executing the Alpha instruction set. It consists of a 500 MHz pipelined datapath with hardware support for floating-point operations. The pipeline has 8 stages: instruction fetch, register-read, ALU 1 through 5, and write-back. The 5-stage ALU supports pipelined floating-point and multiply instructions. However, most instructions execute in a single cycle. The PC 106 includes several performance enhancing features including a branch target buffer, pre-compute logic for branch conditions, and a fully bypassed datapath. The PC 106 interfaces with separate first-level instruction and data caches designed for single-cycle latency.

Detailed Description Text (23):
As will be described in more detail below, the system uses 64 KB two-way set-associative, blocking caches with virtual indices and physical tags. The L1 cache modules 108, 110 include tag compare logic, instruction and data translation lookaside buffers (TLBs) (each storing 256 entries, in a 4-way associative caching arrangement), and a store buffer (data cache only). The L1 cache modules 108, 110 also maintains a 2-bit state field per cache line, corresponding to the four states in a typical MESI protocol. For simplicity, the L1 instruction cache modules 108 and L1 data cache modules 110 use virtually the same design. Therefore, unlike other Alpha implementations, the instruction cache is kept-coherent by hardware. Treating all cache modules 108, 110 in the same way also simplifies the implementation of a no-inclusion policy at the L2 level.

Detailed Description Text (25):
Referring to FIG. 3, conceptually, the ICS 112 is a crossbar that inter-connects most of the modules 150 on a processor node 102 or I/O node 104. The ICS 112 includes a switch fabric 152 and an arbiter 154 for determining which data transfer(s) to handle during each available data transfer period. The length of the data period depends on the number of transfers required to send one cache line across the ICS 112. In a preferred embodiment, each connection provided by the switch fabric 152 of the ICS 112 has a path width of 64 data bits, plus eight parity bits, for a total of 72 bits. Each cache line transported through the ICS 112 has 512 bits of data and sixty-four parity bits. Memory lines are transported along with the corresponding sixty-four parity bits when they are transported through the ICS 112. Parity bits for memory lines are also sent to and used in the L1 cache arrays. However, parity bits are not used in the L2 cache and they are also not used in main memory. Instead, in the L2 cache, 20 ECC bits are associated with each memory line, and more specifically a 10-bit ECC is associated with each 256-bit half memory line. In the L2 cache and main memory, the 64 bits otherwise available for use as parity bits are used instead to store the 20 ECC bits, as well as a 44-bit directory entry, which will be described in more detail below. Data transfers generally are sent with a command or transaction type indicator, which is transferred in parallel with the first 64 bits of data of the cache line. Each cache line sized data transfer requires eight clock cycles, with 64 bits of data and a proportional share of the parity and ECC bits being transferred during each clock cycle.

Detailed Description Text (26):
Arbitration and flow control are handled by the arbiter 154. To better understand the arbiter it is helpful to first review the interface 156 presented by each module 150 (i.e., L1 cache modules 108, 110, L2 cache, protocol engine or system controller) to the ICS 112. As shown in FIG. 3, the standard intra-chip interface 156 provided by each such module includes one or more input buffers 160, one or more output buffers 162, a first finite state machine (FSM) 164 for controlling use of the input buffer(s) 160, and a second finite state machine (FSM) 166 for

controlling use of the output buffer(s) 162. The arbiter 154, via the FSM 164, 166 of each module 150 keeps track of the availability of buffer space in the output buffers 162 of the modules 150 at all times, and exercises flow control by deferring requests to transfer data to modules with full input buffers 160. The arbiter 154 also receives all intra-chip data transfer requests from the interfaces 156 of the modules 150, and arbitrates between the requests whose destinations have input buffers 160 with sufficient room to receive a data transfer (i.e., a cache line of data).

Detailed Description Text (27):
In a preferred embodiment three parallel communication lanes, also called queues, are implemented in the input buffers 160 and output buffers 162 of the ICS interface 156, as well as in the input and output buffers of interfaces (not shown) to the packet switch 126 and interconnect 134 (see FIG. 1). These lanes or queues are labeled I/O, low priority and high priority, respectively. The high priority queues in the input and output buffers are used to store messages sent from a home node to another node of the system, replies from third party nodes to the home node or the requester node for a particular transaction, and messages internal to a node. The low priority queues are used to store messages going to the home node for a particular transaction. The low priority message are thus messages for initiating new memory transactions, while the high priority messages are messages for completing previously initiated memory transactions. The I/O queues are used for handling requests being sent to I/O devices. The messages in the I/O queues are given the lowest priority by the intrachip switch 112 and also by the packet switch 126 and interconnect 134 (see FIG. 1).

Detailed Description Text (28):
The use of multiple communication lanes generally increases the size of the input and output buffers in the interfaces to the ICS 112, packet switch 126 and interconnect 134. However, the use of multiple communication lanes is important for avoid deadlock conditions in the network, and in particular for ensuring that active memory transactions make forward progress even when the system is experiencing high levels of protocol message traffic. In alternate embodiments, four or more communication lanes are used instead of three. In particular, in one alternate embodiment the high priority lane is replaced by two separate communication lanes, one for messages sent from the home node of a memory transaction and the other for replies sent by third parties to either the home node or any other node in the system. Providing the additional communication lane helps to ensure that messages sent by the home nodes of transactions are not blocked by reply messages being sent by the same node(s) for transactions in which those nodes are not the home node, and vice versa.

Detailed Description Text (30):
The ICS 112 uses a unidirectional, push-only data transfer technique. The initiator of a memory transaction always sources data. If the destination of a transaction is ready, the arbiter 154 schedules the data transfer according to datapath availability. A grant is issued by the arbiter 154 to the initiator of the transaction to commence the data transfer at a rate of one 64-bit word per cycle without any further flow control. Concurrently, the destination receives a signal from the arbiter 154 that identifies the initiator and the type of transfer. Transfers across the ICS 112 are atomic operations.

Detailed Description Text (47):
Each of the protocol engines 122, 124, as shown in FIG. 5, includes an input controller 190, preferably implemented as a finite state machine used in connection with a set of input buffers 192 for receiving data (inbound messages) from the ICS 112 and the PS 132. Received messages, some of which include a full cache line of data and the associated parity bits, are stored in the input buffers 192. In a preferred embodiment, sufficient input buffers 192 are provided to store inbound, received data for up to sixteen ongoing memory transactions. A test and execution unit 194 (herein called the execution unit) executes instructions obtained from an instruction memory 196, also called the microcode array, so as to advance memory transactions, also called cache coherence transactions. The currently selected instruction, obtained from the instruction memory 196, is held in a current instruction buffer 197 for decoding and execution by the execution unit 194. Output messages generated by the execution unit 194 are stored in a output buffers 198 the operation of which are controlled by an output controller 200, preferably implemented as a finite state machine. The output messages are transferred from the output buffers 198 to specified destinations within the same node 102, 104 as a protocol engine 122, 124 via the ICS 112 or to specified destinations within other nodes 102, 104 of the multiprocessor system 100 via the PS 132.

Detailed Description Text (49):
FIG. 6A shows the format of each of the instructions stored in the instruction memory 196 and instruction buffer 197. As shown, each instruction includes an operator, two operands, and a next program counter field. The operator indicates the type of operation to be performed by the execution unit 194 when executing the instruction, the two operands provide parameters that affect the execution of an instruction.

Detailed Description Text (50):
The current state of multiple memory transactions is stored in a set of registers collectively called the Transient State Register File (TSRF) 202. Each memory transaction has a memory line address (sometimes called the global memory address) that identifies the memory line that is the subject of the memory transaction. More specifically, the memory line address identifies the node 102, 104 that interfaces with the memory subsystem 123 that stores the memory line of information 184 (i.e., home node) and a specific position within the memory subsystem 123 of the memory line of information 184. In a preferred embodiment, the top M (e.g., 10) bits of the memory line address identify the home node 102, 104 of the memory line of information 184, while the remainder of the address bits identify the memory line 184 within the identified node. In a preferred embodiment, the memory line address for a memory line does not include any of the address bits used to identify sub-portions of the memory line, such as individual 64-bit words of individual bytes within the memory line of information 184. However, in other embodiments that support transactions on sub-portions of memory lines, the memory line addresses used may include bits for identifying such memory line sub-portions.

Detailed Description Text (51):
Referring to FIG. 6B, each memory transaction has a respective entry 210 stored in the Transient State Register File (TSRF) 202 that indicates the state of the memory transaction. In a preferred embodiment, the TSRF 202 has registers for storing sixteen entries 210 as well as access circuitry for reading and updating the contents of the TSRF entries 210. Obviously the number of entries in the TSRF 202 is a design choice that will vary from one implementation to another. Typically, the TSRF 202 will include at least as many entries as the number of PCs 106 included in a processor node 102.

Detailed Description Text (53):
Referring to FIGS. 6B, 7A-7C, and 8, the sequence of operations required to execute an instruction so as to advance a memory transaction is: reading the TSRF entries, scheduling one of the transactions represented by the TSRF entries, retrieving from the instruction memory the instruction identified by the TSRF of the scheduled transaction, and executing the instruction. As shown in FIGS. 7A and 7B, this sequence of four operations is pipelined and is furthermore performed by two "logical pipelines" that are parallel but offset from each other by one clock cycle. One logical pipeline is for the odd TSRF entries and the other is for the even TSRF entries. However, the two logical pipelines are implemented using a shared scheduler 212, a shared microcode array 196 and access circuitry (see FIG. 8), and shared execute logic 240, which along with the scheduler 212 is part of the test and execution unit 194. Only the TSRF registers and access circuitry 202 have distinct even and odd circuits.

Detailed Description Text (71):
a set of counter fields 226: are used to store count values that, for example, control repeated execution of an instruction (e.g., when a transaction needs to send out N identical protocol messages to other nodes 102, 104, one of the counter fields 226 is initially to a value corresponding to N, and is then decremented or incremented after each execution of the instruction until a predefined terminal count value is reached, at which point the memory transaction is either complete or a next program counter for the transaction is determined). The counter fields 226 and the state field 220 together form an overall or more specific state of an associated memory transaction.

Detailed Description Text (81):
The scheduler 212 includes arbitration logic for selecting the next even TSRF entry and the next odd TSRF entry to be sent to the execution unit 194 in accordance with (A) the states of the TSRF entries, (B) the buffered received messages received via the PS 132 and the ICS 112 and which TSRF entry, if any, corresponds to each of the buffered received messages, and (C) a set of prioritization rules. Each TSRF entry and each buffered received message identifies the memory line associated therewith, and the arbitration logic of the scheduler includes an array

of comparators for comparing the memory line addresses in the TSRF entries with the memory line addresses in the buffered received messages so as to produce a corresponding set of status update signals. The status update signals are used for "upgrading" TSRF entries from the Waiting and Local_Waiting state to the active state, as well as for downgrading the TSRF entry for the last running transaction to the waiting, local waiting or vacant state, depending on whether the transaction is finished, and if not finished, what type of message (i.e., from the local node or a remote note) the transaction needs to receive in order to ready to resume execution.

Detailed Description Text (82):
The status update signals are also used to determine when a buffered received message has the same address as a previously allocated TSRF, but is for a different memory transaction. When this condition is detected by the arbitration logic, one of three actions is performed: (A) a new TSRF entry is allocated for the transaction associated with the received message, and the new transaction is suspended, (B) the received message is merged into previously allocated transaction and modifies its state, or (C) the message is temporarily left in the input buffer because the previously allocated transaction is not currently in a state allowing the received message to be merged with it, and the received message is then either merged with the previously allocated transaction or, if that transaction completes, a new TSRF is allocated for the new message and that TSRF is placed in the Active state. When the received message is of the type that could potentially be merged with a previously allocated transaction, the previously allocated transaction must be in the Waiting or Local_Waiting state before the merger can be performed. When a Receive instruction is executed, the transaction enters a Waiting or Local_Waiting state. The transaction can not enter the Active state until either (A) one of the predefined messages required to advance the transaction, or (B) one of the predefined messages that can be merged with the transaction is received.

Detailed Description Text (83):
Referring to FIGS. 6B and 8, the scheduler 212 selects between continued execution of the currently Running transaction and any of the other Active transactions, if any. FIG. 6B shows a portion of the logic for selecting an Active transaction. FIG. 8 shows logic for continuing execution of a currently Running transaction. On the right side of FIG. 8 is shown a current instruction buffer 197 for holding the current instruction for Running transaction.

Detailed Description Text (84):
The operator and arguments of the current instruction are passed to the execute logic 242, which also has access to all the fields of the TSRF of the Running transaction. The execute logic computes a set of condition codes, labeled "Curr_CC" in FIG. 8, as well as new State and Next PC for the TSRF of the running transaction. The Next PC, to be stored in the TSRF of the current Running transaction, is obtained from the current instruction stored in buffer 197. The execute logic 242 may also update one or more counters in the TSRF of the current Running transaction as well as other fields of the TSRF.

Detailed Description Text (85):
When the scheduler 212 determines that the current Running transaction should continue to run, the next instruction for the transaction is determined as follows. The current instruction in buffer 197 includes a "Next PC" field that specifies the base address of a next instruction. Predefined bits (e.g., the four least significant bits) of the "Next PC" address are logically combined (by logic gate or gates 244) with the condition codes (Curr_CC) generated by the execute logic 242 so as to generate a microcode address that is stored in microcode address latch 246. Multiplexers 248 and 250 are provided to facilitate selection between the current Running transaction and another Active transaction. Multiplexers 248 and 250 operate during both Even and Odd clock cycles so as to perform separate instruction retrieval operations during Even and Odd clock cycles (See FIG. 7A).

Detailed Description Text (100):
The L1 cache receives data from and sends data to the L2 cache, main memory, and other devices via the intra-chip switch 112. Received data is temporarily buffered by a data in buffer 310, and data being sent elsewhere is output via an output finite state machine (Output FSM) 312. The output buffer for sourcing data to the ICS 112 is called the Fwd/Evt buffer 366.

Detailed Description Text (101):
Input logic 314 receives control signals sent via the ICS 112 and conveys those control signals

to either a fill FSM 316 or a synonym FSM 318. The fill FSM 316 controls the loading of a cache line received from the ICS 112 into the L1 cache data array 262. The synonym FSM 318 controls the movement of a cache line from one L1 cache slot to another when the L2 cache instructs the L1 cache to do so. Multiplexer 320 routes cached data from a slot of the L1 cache data array 262 back to the data array input multiplexer 322 under the control of the synonym FSM 318. Input and output staging buffers 321, 323 are preferably used in this data path, for instance to facilitate delivery of successive portions of the data in a cache line over the data path.

Detailed Description Text (102):
When the synonym FSM 318 is not active, multiplexer 320 sources data from the data input buffer 310 to the data array input multiplexer 322. The movement of a cache line from one L1 cache slot to another is required when the cache line index derived from a virtual address does not match the physical location of a cache line in the L1 cache. A tag information input multiplexer 324 is also controlled by the synonym FSM 318 to enable tag information for the L1 tag array 274 to be sourced by synonym information from the synonym FSM 318 when the synonym FSM 318 is activated. When the synonym FSM 318 is not activated, the tag information input multiplexer 324 sources tag information for the L1 tag array 274 from the virtual address (PC_vaddr) provided by the processor core.

Detailed Description Text (104):
When the processor core sends either a read or write request to the L1 cache, the processor core provides a virtual address, PC_vaddr. The virtual address and information derived from it, such as a valid tag match signal, are stored in a series of staging buffers 332, 334, 336. Additional staging buffers, beyond those shown in FIG. 9C, may be required in some implementations. The virtual address is translated into a physical address (PA) by a translation lookaside buffer (TLB) 340 at the same time that a tag and state lookup is performed by the tag and state arrays 274, 266. The resulting physical address and tag lookup results are stored in a second staging buffer 334 and are then conveyed to a tag checking circuit 342 that determines if there is a tag match for a valid cache line. The results of the tag check, which includes state information as well as tag match information and the virtual address being checked, are stored in yet another staging buffer 336. The information in the staging buffer 336 is conveyed to a data write FSM 360 when a valid match is found, and is conveyed to the output FSM 312 when a cache miss is detected. The final staging buffer 336 also stores a "replay" signal, generated by the tag checking circuit 342, and the replay signal is conveyed to the processor core to indicate whether the L1 read or write operation requested by the processor core must be resubmitted to the L1 cache after the PC_inhibit signal is deactivated.

Detailed Description Text (105):
When a data write is being performed, the write request signal (PC_WrRq) and the results of the tag lookup are used by a data write FSM 360 and a cache access Arbiter 362 to determine if (and when) the data sourced by the processor core is to be written into the L1 cache data array 262. The data sourced by the processor core is buffered in a series of staging buffers 352, 354, 356 so that the data to be written is available at the data array input multiplexer 322 at the same time that the tag check results become available to the data write FSM 360. The data write FSM 360 stalls the data pipeline 352, 354, 356 if the arbiter 362 determines that the L1 cache is not ready to store the sourced data into the L1 cache data array 262.

Detailed Description Text (106):
When a data read is being performed, the read request signal (PC_RdRq) is received directly by the arbiter 362 and the virtual address is used to directly read a cache line in the data array 262 even before the results of the tag lookup and check are ready. The data read from the data array is temporarily buffered in staging buffer 321 and is discarded if a cache miss is detected. If the read data is being read in response to a processor core request and a cache hit is detected, the read data is sourced from the staging buffer 321 to the processor core via the data path labeled Array_Out Data (L1_PC_data). If the read data is being read in response to a request received via the ICS 112, the read data is sourced from the staging buffer 321 to the Fwd/Evt buffer 366, and from there it is conveyed to the output FSM 312 for transmission to the requesting device via the ICS 112.

Detailed Description Text (121):
FIG. 10C shows the data paths and primary components of the L2 cache 116. As described earlier with respect to FIG. 3, the L2 cache has an interface to the intra-chip switch 112. This

interface includes one or more input buffers 160, one or more output buffers 162, an input finite state machine (In FSM) 164 for controlling use of the input buffer(s) 160, and an output finite state machine (Out FSM) 166 for controlling use of the output buffer(s) 162. Similarly, the L2 cache 116 has an interface to the memory controller 118 (see also FIG. 1) that includes one or more input buffers 400, one or more output buffers 402 and a memory controller interface finite state machine (MC interface FSM) 404 for controlling the use of the MC interface input and output buffers 400, 402.

Detailed Description Text (122):
A set of pending buffers 406 are used to store status information about memory transactions pending in the L2 cache. For instance, the pending buffers 406 keep track of requests made to the memory subsystem (see FIG. 1) via the memory controller 118. A set of temporary data buffers 408 are used to temporarily store cache line data associated with pending memory transactions, including data being sourced to the L2 cache, data sourced from the L2 cache, and data transported through the L2 cache (i.e., from the memory subsystem 123 to the L1 cache). Data sent by the L2 cache in response to an L1 cache miss bypasses the temporary data buffers 408 and is sent via a bypass data path 410 so as to reduce latency when the L2 cache contains the data needed to satisfy a cache miss in an L1 cache (which is coupled to the L2 cache via the ICS 112).

Detailed Description Text (124):
When an L1 cache miss is serviced by the L2 cache 116, and the L2 cache does not have a cached copy of the memory line required by the L1 cache, the request is forwarded to the memory subsystem 123 via the MC interface FSM 404. The memory line of information provided by the reply from the memory subsystem 123 is not stored in the L2 cache 116. Instead the memory line is sent directly to the L1 cache, bypassing the L2 data array 292. More specifically, the reply from the memory subsystem is directed through multiplexer 414 to the Din2 input port of the temporary data buffers 408. The reply is then output at the Dout1 port of the temporary data buffers 408 to the interface output buffer 162 via output multiplexer 416.

Detailed Description Text (125):
When an L1 cache evicts a memory line from the L1 cache, the victim memory line is sent to the L2 cache for storage via the ICS 112 and the interface input buffer 160. The victim memory line is received at the Din1 input port of the temporary data buffers 408 and temporarily stored therein. The victim memory line is then sent from the temporary data buffers 408 to the L2 data array 292, via the Dout2 port of the temporary data buffers 408 and a staging buffer 418, for storage in the L2 data array 292.

Detailed Description Text (126):
When the L2 cache sources a memory line to an L1 cache, the memory line read from the L2 data array 292 is conveyed via bypass line 410 to output multiplexer 416, and from there to the ICS interface output buffer 162. The output FSM 166 handles the transfer of the memory line from the output buffer 162 to the ICS 112, and from there it is sent to the L1 cache.

Detailed Description Text (153):
Upon receiving the read-exclusive response (step 1252), the steps taken depend upon the content of the response. As noted above, a read-exclusive request can result in a number of invalidation acknowledgments from nodes 102, 104 that have or had a shared copy of the memory line of information 184. Additionally, the CCP does not requires protocol message ordering, so invalidation acknowledgments can arrive at the requesting node before a read-exclusive reply. If the response is an invalidation acknowledgment (step 1253-Yes), RPE 124 updates the TSRF entry 210 in the TSRF 202 associated with this memory transaction to reflect that the invalidation acknowledgment was received (step 1256). More specifically, RPE 124 increments or decrements a counter in the counter fields 226 of the TSRF entry 210.

Current US Cross Reference Classification (1):
707/10

Current US Cross Reference Classification (2):
707/201

Previous Doc Next Doc Go to Doc#

L5: Entry 6 of 8 File: USPT Feb 12, 1985

DOCUMENT-IDENTIFIER: US 4499539 A
TITLE: Method and apparatus for limiting allocated data-storage space in a data-storage unit

Abstract Text (1):
A data-storage buffer transfers data signals with other units in relatively large blocks of data. Such large blocks storable in large address spaces are not always filled with meaningful data. To more efficiently use the data-storage space in the data-storage buffer, the allocatable unit or segment of the data buffer is made smaller than the data capacity of the large block. Each time a large block of data is to be written into the data buffer, a sufficient number of the segments for storing data of one large block is allocated for receiving the data. After the data of the one block is written into the data buffer, the allocated segments are examined; all of the allocated segments not storing data from the one large block are deallocated. The invention is particularly useful for data buffers acting as cached data storage for large-capacity direct-access storage devices (DASD) and are coupled to host processors programmed to operate with such DASD. The procedure is followed for data written into the caching data storage whether supplied by DASD or the host processors.

Brief Summary Text (8):
The management of data-storage apparatus for ensuring full utilization of such space available in any data storage unit includes storing variable-length data. For example, U.S. Pat. No. 3,739,352, shows a microprogrammed processor associated with a so-called "free-field" memory in which operands of any length in terms of number of bits can be processed. The free-field memory is addressed by an address register that points to the boundary between any two bits stored in the memory as the start of a field and indicates the number of bits in the field up to a maximum bit capacity of the memory. While this technique certainly appears to provide for a maximal packing of a given memory (data-storage unit), when such data is replaced by other data the probability of the replacing data having an extent (number of bits) equal to the data being replaced is relatively small. This means that each time data is replaced that the memory must be reformatted if the storage efficiency is to be maintained. Accordingly, this technique, while probably valuable for many applications, is not applicable to a front-store/back-store data-storage hierarchy because of the data replacement operations. As a result of such a scheme, it can be easily envisioned that fragmentation of data would occur which requires extensive and time-consuming management techniques not desired in a peripheral data-storage hierarchy.

Brief Summary Text (12):
In a peripheral data-storage unit, U.S. Pat. No. 4,103,329 shows handling data represented by variable field length for using less data-storage. The bit fields are handled independently in the natural storage addressing elements and boundaries. This patent shows initializing a displacement register to contain an element displacement from a base address which contains the first bit of a desired bit field. While such a technique is certainly appropriate for packing data into a main memory for use by a host processor, the complexity and tracking of all of such data wherein the quantity of data is in the megabyte range becomes excessively expensive. Accordingly, these later-described techniques are also not fully satisfactory for managing a front store of a peripheral data-storage hierarchy.

Brief Summary Text (16):
In accordance with the invention, data-storage apparatus has allocatable data-storage spaces each having a data-storage capacity less than a maximal capacity required for given data transfers. Upon each incoming data transfer, an initial maximal allocation of data-storage space in the data-storage unit is made for the expected data. The data is then transferred to the allocated data-storage space. Upon completion of the data transfer, an examination is made of which allocated data-storage spaces actually received none of the incoming data. All of

those allocated data-storage spaces receiving no such data are then deallocated and allowed to be used for storing other data.

Brief Summary Text (17):
In another context, a maximal initial allocation of data storage space is made in accordance with the size of a maximal data transfer and upon completion of the data transfer the initial allocation is reduced to the size of the actual data transfer with all remaining portions of the allocated data-storage spaces being deallocated.

Detailed Description Text (3):
Directory 16 has a plurality of registers, as later detailed with respect to FIG. 5; each of which includes a stored DASD address (DASD ADDR) 20 which identifies the DASD 14 address which is intended to store or is actually storing data that is currently stored at an addressable portion of cache 15. The cache 15 storage location is indicated by cache address pointer P1 contained in section 22 of each directory 16 register or can be indicated by which directory 16 register is storing DASD ADDR; i.e. directory 16 register addresses are mapped to cache 15 addressable data-storage spaces. In accordance with the invention, cache 15 has allocatable data-storage spaces (sets of addressable data-storage register which are allocated as a single unit) having a capacity smaller than the data-storage capacity of a data-storage track 21 on DASD 14 pointed to by the DASD address 20 in directory 16. Preferably the capacity of the allocatable data-storage space in cache 15 is an integral submultiple of the maximum capacity of the data-storage track 21. For the present description the allocatable data-storage space of cache 15 is one-third the capacity of the DASD 14 track capacity. Three cache 15 addresses are required to address contents of a complete DASD track stored in cache 15. This addressing is achieved by having an address pointer P1 in section 22 of directory 16 identifying an allocatable data-storage space 23 of cache 15 for storing a first one-third portion of a DASD 14 track. In this one portion of allocatable data-storage space 23 are a pair of pointers P2, P3, respectively in spaces 24 and 26. P2 contains an address pointing to allocatable data-storage space 25 of cache 15 which stores the second one-third portion of track 14 while P3 points to an addressable, allocatable data-storage space 27 of cache 15 which stores the third one-third portion of track 14. In this manner the three allocatable data-storage spaces of cache 15 are concatenated to store the data contents of one track. It is to be understood that the number of said multiples of allocatable units in cache 15 may be a number other than three and that the additional pointers such as pointers P2 and P3 instead of being stored in the first allocatable space 23 pointed to by directory 16 may in fact be stored in area 22 along with pointer P1 in directory 16. Storing the addresses P2, P3 within cache 15 simplifies directory 16's structure; it does require one additional access to cache 15 for setting up data transfers, as will become apparent.

Detailed Description Text (4):
In accordance with the invention, when cache 15 is to receive data from either a using unit 10 or from DASD 14, programmed processor 17 will not have an indication of the extent of data which cache 15 will receive. That is, it may be a full track of data or less than a full track of data. Accordingly, each time cache 15 is to receive data, three allocatable data-storage spaces of cache 15 are allocated for the upcoming data transfer. Following such allocation, the data transfer ensues. Upon completion of the data transfer, programmed processor 17 examines which of the allocated data-storage spaces, such as 23, 25 and 27, have in fact received data for storage. Those allocated data-storage spaces of cache 15 not receiving any data during such data transfer are then deallocated, via appropriate pointers being zeroed and are made available for reallocation of data not related to the just-addressed DASD 14 track. In this manner, management of cache 15 data-storage space allows a greater number of tracks to be effectively stored in cache 15 with a relatively smaller cache capacity, i.e. reduces cost. In another view, if the same size cache 15 is used, then a greater performance is provided because the contents of a greater number of addressable DASD 14 data-storage tracks can be stored in cache 15. Cache 15, of course, has a large plurality of such allocatable data-storage spaces, as collectively indicated by numeral 28 and ellipsis 29.

Detailed Description Text (5):
The scattering of data from a single addressable data-storage track into a plurality of unrelated segments or data-storage spaces of cache 15 when high-speed data transfers are involved requires rapid concatenation and some buffering during the data transfer. For enhanced flow of data signals into and out of cache 15, as best seen in FIGS. 1 and 2, a plurality of system storageaddress registers 30 are provided. SSAR-0 receives pointer P1 from directory 16

in preparation for accessing allocated data-storage space 23. SSAR-1 and SSAR-2 respectively receive pointers P2 and P3 from areas 24, 26 of allocated data-storage space 23. This action completes the preparation for an ensuing data transfer. FIG. 4 shows how a plurality of address registers can quickly concatenate a plurality of addressable data-storage spaces for receiving a highspeed burst of data signals.

Detailed Description Text (6):
Upon each received data-access request, programmed processor 17 examines directory 16 to determine whether or not an associated allocatable data-storage space has been allocated to the DASD address received from using unit 10. If there is no match, then a cache-miss occurs, as indicated by numeral 35. Such a cache-miss can result in a data promotion from DASD 14 into cache 15, as will be later detailed. Such a miss activates processing unit 19 to access control store 18 for executing program 36 which may result in transferring data from DASD 14 to cache 15. Further, a cache write-hit, which indicates data will be transferred from a using unit 10 into cache 15, results in programmed processor 17 responding, as indicated by arrow 37, to use program 36 for preparing cache 15 to receive data from host 10, which may include up to a full track of data the extent of which, of course, is not presently known to the data-storage system. In such a host write, the host data is preferably simultaneously written to DASD 14.

Detailed Description Text (7):
In any event, for cache 15 to receive data without any overrun exposure results in programmed processor 17 in responding to program 36 to execute program 40 for allocating one DASD track capacity in cache 15 and setting the pointers P1, P2 and P3 as may be required. For example, if no cache 15 data-storage space has been allocated, then three cache 15 data-storage spaces are allocated with the corresponding pointers being generated. If on the other hand only one data-storage space is currently allocated, then two more data-storage spaces are allocated with the corresponding pointers being generated, all of which is detailed later with respect to FIG. 3. Upon completion of executing program 40, programmed processor 17 executes program 41 which actually causes the transfer of data to the cache 15 from either host 10 or DASD 14, as the case may be. Upon completion of the data transfer, programmed processor 17 checks the ending address of the last byte of data transferred into cache 15 by executing program 42. This check identifies which allocated data-storage spaces in fact received no data during the data transfer; i.e. the ending address check determines which of the three allocated data-storage space last received data. Then programmed processor 17 by executing program 43 deallocates any unused cache allocations made for the data transfer. Of course, preparatory to the execution of programs 36 and 43 and thereafter, other programs 44 which are commonly found in data-storage subsystems are executed. Since such programs do not have a bearing on an understanding of the present invention, they are not detailed.

Detailed Description Text (8):
In program 40 it may be required that programmed processor 17 replaced existing data in cache 15. Free or unallocated data-storage spaces must be identified. This action is achieved by an LRU (least recently used) replacement control list 47, usually found in data-storage hierarchies, explained with respect to FIG. 5. LRU 47 includes identification of those allocatable data-storage spaces which are available for allocation. Accordingly, execution of program 40 by programmed processor 17 results in usage of MRU-LRU program 46 for scanning LRU 47 to allocate data-storage spaces. If sufficient allocatable data-storage spaces are found, then those spaces are allocated with no further activity. However, if no allocatable data-storage spaces are found, then programmed processor 17 uses replace program 45 for transferring data from a replaced one of the allocated data-storage spaces to DASD 14 using known replacement techniques. When DASD 14 is updated concurrently with cache 15, the cache 15 space is immediately reallocated to the incoming data without any prereplacing data transfers to DASD 14. In this manner, cache 15 can be always filled with promoted data. Program 48 enables programmed processor 17 to access cache 15 using known techniques; accordingly, this program is not detailed.

Detailed Description Text (9):
FIG. 2 illustrates a preferred embodiment of the invention as employed in a two-storage director 12 data-storage arrangement. Each storage director 12 includes a plurality of so-called channel adaptors 50, also separately denominated as CAA through CAH, which connect the respective storage directors 12 to a plurality of using units 10 via a plurality of input/output connections 11. Each storage director 12 includes a programmed processor 17 which, as usual, includes a processing unit 19 having a control store 18 which contains computer

programs for performing the storagedirector functions. FIG. 2 shows the logical structure; i.e. the functions performed by processor 19 in executing the programs in control store 18. The programmed processor 17 includes programs constituting address and command evaluator ACE 52 which receive and evaluate using unit 10 supplied peripheral commands. Such functions are also performed in present day storage directors for noncached DASD as widely sold throughout the world and are a part of other programs 44 in FIG. 1. The programmed processor 17 also includes programs for direct access control DAC 53 which responds to commands evaluated and decoded by ACE 52 to control data transfers between using units 10 and DASD 14, as well as providing device commands to DASD 14 for performing well known DASD access and control functions. DAC 53 includes program 41 as well as programs for accessing DASD 14 included in other programs 44 relating to accessing DASDs 14 and transferring data between using units 10 and DASDs 14, all of which is well known. Programmed processor 17 further includes programs CAC 54 which is a cache access control for accessing cache 15. CD latches 59, one for each of the DASDs 14, are accessed by DAC 53 and CAC 54 respectively for determining whether to access cache 15 or DASD 14 directly and for setting the latches to D upon a cache miss. Connections from storage director 12 to DASDs 14 are via DASD circuits 55 which are constructed using known device adaptor and data-flow design techniques. Cache 15 is accessed via memory circuits (MEM CCTS) 56 which includes those circuits for generating addresses and access requests including SSARs 30. Cache 15 is a portion of a large random-access store 57, hereinafter referred to as a system store. It is preferred that cache 15 can simultaneously and independently handle data transfers with a DASD 14 and a host 10. The directory 16 and LRU 47 for cache 15 are also stored in system store 57. Additionally, any using unit 10 can command the storage directors 12 to keep data in cache, i.e. pin or bind the data to cache 15. For all bound tracks, it records a cache bound list 60, stored within directory 58 but shown separately for clarity, indicates to both storage directors 12 which data stored in cache 15 is to remain in cache 15. Such bound data is not listed in LRU 47 for preventing replace program 45 from reallocating cache 15 space.

Detailed Description Text (11):
Programmed processor 17 at 70 receives a storage-access request. This request is decoded and evaluated in ACE 52 using known techniques. At 71, programmed processor 17 DAC 53 portion examines the CD latch 59 (FIG. 2) related to the DASD 14 addressed in the received storage-access request to determine whether cache 15 or only DASD 14 to the exclusion of cache 15 is to be accessed. For a direct access, DASD 14 is accessed at 72 using usual DASD access methods. For a cache C access, programmed processor 17 searches directory 16 at 73 to determine whether or not the track requested in the received storage-access request (I/O command) has allocated space in cache 15. In this regard it is noted that some commands will require a direct connection to DASD 14 to the exclusion of cache 15. Accordingly, ACE 52 in detecting such a received I/O command sets latch 59 for the addressed DASD 14 to the direct mode "D". An example of such an I/O command is to recalibrate a DASD 14. Searches and SEARCH ID EQUAL commands can be performed for cache 15 accesses within directory 16, i.e. the commands are performed in a virtual manner not involving DASD 14. In the preferred embodiment directory 16 does not separately identify records in a track; only tracks are identified, no limitation thereto intended. Upon completion of the directory 16 search, programmed processor 17 at 74 determines whether or not a cache-hit has occurred. If a cache-hit occurred, which is preferred programmed processor 17 at 74A transfers the P1 stored in section 22 of the directory 16 register identified by the received DASD 14 address to SSAR-0; then it transfers P2 and P3 respectively to SSAR-1 and SSAR-2 from their respective storage locations.

Detailed Description Text (12):
At step 75, director 12 examines cache 15 to determine whether or not the record to be accessed is stored in cache 15 (record hit). If the addressed record is in cache 15 (record hit is yes), then additional segments may not be needed to successfully complete the ensuing data transfer. Then at step 76 the type of data transfer operation to be performed is examined. For a read operation R (transfer of data to a host 10), director 12 at step 77 transfers the requested data from cache 15 to the requesting host 10. Such transfer completes the operation permitting director 12 to exit the machine operation at 78 for performing other data processing operations. For a write operation W (transfer of data from a host 10) indicated at step 76, director 12 at step 80 examines the received host 10 supplied command for ascertaining if the write is a FORMAT write (access to DASD 14 is requested to the exclusion of cache 15) or any other form of write (cache 15 is to be utilized) is requested. For a FORMAT write, director 12 in step 81 deallocates any allocated cache 15 data-storage space and transfers the received data to DASD 14. For a nonformat write (FORMAT=0) at step 80, director 12 in step 82 transfers data from the requesting host 10 to both cache 15 and DASD 14 respective addressed data-storage

areas. In this manner, cache 15 and DASD 14 always have identical copies of the same data. From steps 81 and 82, director 12 proceeds to other data processing operations via logic path 78. This operation allows less than a full track allocation in cache to handle successive data transfers (partial track allocations).

Detailed Description Text (13):
Returning to step 75, when director 12 does not find the addressed record (record hit is no), then for the impending data transfer to the cache, additional segments may be allocated for the ensuing data transfer. In steps 85 and 86 director 12 examines the values of P2 and P3. For either or both pointers being zero (no corresponding space has been allocated in cache 15), director 12 in step 86 allocates an additional segment to the track, as previously described and then proceeds to transfer data to cache 15 at 87. The step 87 data transfer can be a write from a host 10 to DASD 14 and cache 15, a read from DASD 14 to cache 15 and a host 10, or a staging data operation from DASD 14 to cache 15.

Detailed Description Text (14):
The post-transfer machine operations find director 12 examining cache 15 to determine which of the three allocated segments in fact received data from the just-completed data transfer. In step 88, director 12 examines a later-described "k-counter" 129 (FIG. 4) to ascertain the values 1, 2 or 3 which respectively indicate that one, two or three allocated segments (corresponding to P1, P2, P3) in fact received and are currently storing data. For k=1, director 12 in step 90 takes the segments 2 and 3 (also termed XM and YM, respectively) identifications and inserts same into the LRU list for making these segments available for allocation. In step 91, the corresponding pointers P2 and P3 are set to zero. For a value k=2 in step 88, director 12 in steps 92 and 93 inserts the third segment YM into the LRU list and sets pointer P3 to zero. For a value of k=3 in step 88, director 12 knows that all three segments have received and are currently storing data, hence it proceeds directly to do other data processing operations through logic path 78. Path 78 is also entered from steps 91 and 93, as well.

Detailed Description Text (15):
For a cache miss at 74 (hit=0), director 12 in steps 95 and 96 allocates three segments in cache 15 (XY, XM, YM) for the ensuing data transfer to cache 15 and sets the corresponding pointers P1, P2 and P3 in the respective SSAR's 0, 1 and 2. Then director 12 proceeds to the data transfer operation performed in step 87, as previously described.

Detailed Description Text (16):
In one embodiment of directory 16, each of the registers in directory 16 corresponded to a space 28 in cache 15. Hence, area 22 is dispensed with the register address within directory 16 also indicating (using base plus offset addressing) the beginning address in cache 15 of an associated space. Allocation then consists of inserting the appropriate DASD address in section 20 of such register. Addresses XM and YM respectively become pointers P2 and P3 and are stored in areas 24, 26 of the area 23 corresponding to address XY. Note there are no changes in directory 16 for these last two pointers. In the event that the last above-described directory 16 structures wherein a given register always is associated with a data-storage area of cache 15, then addresses P2 and P3 are inserted in these respective directory 16 registers; the registers for P2 and P3 are then omitted from LRU 47. The above completes setting up the pointers for the ensuing data transfer.

Detailed Description Text (17):
FIG. 4 illustrates cache 15 addressing circuits usable with the present invention. The data paths 100 extend from cache 15 through memory circuits 56 thence to DASD circuits 55 for data transfers with DASD 14. The data paths also extend to the channel adaptors 50 for data transfers with using unit 10, all as shown in FIG. 2. The data transfers between cache 15, adaptors 50 and DASD circuits 55 are under the control of usual automatic data-transfer circuits of known design and of current use in DASD storage systems. Such automatic transfer control circuits are shown as autocontrol 101 in FIG. 4 as being a part of memory circuits 56. Programmed processor 17 supplies a suitable start signal over line 102 to autocontrol 101. The description assumes the storage system address registers 30 have been loaded with the appropriate addresses P1, P2 and P3 received from programmed processor 17 respectively over address busses 110, 111 and 112. Such loading of address registers by a programmed processor is well known. Once autocontrol 101 receives the START signal, it supplies a cache 15 access enabling signal over line 103 to cache 15. As a result, cache 15 will receive addresses as

later described for accessing data-storage registers within the cache. The access-control signal on line 103 will carry an indication of whether the operation is a read-from-cache operation or a write-to-cache operation. Many caches 15 contain known refresh circuits which interleave refresh cycles with data-access cycles. Each time cache 15 transfers a set of data signals over data path 100, it indicates a cycle of operation to autocontrol 101 over line 104. Autocontrol 101 has been preset in a known manner for transferring a given number of data signals between cache 15 and either DASD 14 or host 10. When data signals are being written into cache 15, autocontrol 101 may not know the number of signals to be received. In this instance, a second signal is supplied over start line 102 to turn autocontrol 101 off for removing the signal on line 103. For example, in a host-to-cache data transfer, the host using the IBM 370 interface architecture can send a socalled COMMAND OUT I/O tag signal indicating the end of the data transfer. Such I/O tag signal results in programmed processor 17 sending a second signal on start line 102 to indicate to autocontrol 101 to terminate the data transfer. Termination of the data transfer either internally to autocontrol 101 or to externally received commands is indicated to programmed processor 15 by an END signal supplied over line 105. For each cycle of cache 15 operation, autocontrol 101 emits an address incrementing signal over line 115. The incrementing signal goes to one and only one of the SSARs 30 as selected by an SSAR address received over bus 130 from programmed processor 17. The addressing of a plurality of address registers is well known and not described for that reason. When P1, P2 and P3 are loaded, the SSAR address signals received from programmed processor 17 will select SSAR-0. Decoder 131 decodes the address signal and supplies an AND circuit enabling signal over line 132 to AND circuits 116 and 120; AND circuit 116 passes the address incrementing signal on line 115 to SSAR-0 for incrementing the address contained therein. Decrementing can be used as well. Each time SSAR-0 is incremented, it supplies a set of address signals to AND circuits 120 for transmitting same over the address bus 123 to cache 15 for selecting the next data-storage location within the cache 15 addressed data-storage space for the data transfer. When the SSAR-0 has counted through all of the addressable data-storage locations within one data-storage space 28 of cache 15, it supplies a carry signal over line 126 through OR circuit 125 for incrementing segment counter 129. Segment counter 129, which counts segments having 2.sup.k data-storage locations (k is an integer), has been preset to zero through a reset signal received from programmed processor 17 overline 140. Segment counter 129 supplies a zero signal over line 141 to decoder 131 for passing the received SSAR-0 address signal to decoding circuits resulting in the line 132 and enabling signal. When segment counter 129 is incremented by the SSAR-0 carry signal, it then supplies a one signal over line 142 to decoder 131.

Detailed Description Text (18):
Decoder 131 is of the type that can add one to the received SSAR address such that the line 132 AND-circuit enabling signal is removed and a new AND-circuit enabling signal is supplied over line 133. Such signal enables AND circuits 117 and 121 associated with SSAR-1. AND circuit 117 enables the address-incrementing signal on line 115 to increment SSAR-1 and then to supply address signals through AND circuits 121 to cache 15. In a similar manner SSAR-1 supplies its carry signal over line 127 to also increment segment counter 129 resulting in a two signal being supplied over line 143 to decoder 131. This causes decoder 131 to add two to the received SSAR address resulting in an AND-circuit enabling signal being sent only over line 134 to AND circuits 118 and 122 associated with SSAR-2. SSAR-2 then receives the address-incrementing signal and supplies the cache data-storage location signals to cache 15 for the third data-storage space being addressed in the sequence of operation.

Detailed Description Text (19):
Segment counter 129 is not restricted to counting segments of 2.sup.k sizes. By providing a segment size register, the counter 129 can count segments having any arbitrary size or variably sized segments. For simplicity segment sizes of 2.sup.k are preferred.

Detailed Description Text (20):
It is to be appreciated that a larger plurality of SSARs 30 may be provided, as indicated by ellipsis 147. As such, any three of the larger plurality of storage address registers may be used in sequencing cache 15 operation in accordance with the invention. Accordingly, there are a like greater plurality of AND circuits enabling lines indicated by ellipsis 148. In any event, the first storage address register which receives P1 is selected by programmed processor 17 in the usual manner. Programmed processor 17 then indicates which SSAR received the P1 address which starts a sequence of concatenated addresses within cache 15 for successively-accessed data-storage spaces 28. Accordingly, a variable number of data-storage spaces 28 can be used with a diversity of sizes of address spaces for receiving data signals. For example, if

two types of DASD 14s are attached to the directors 12, two different sizes of data transfer units (data contents of two DASD tracks have different numbers of stored data bits) may be involved. From a first DASD 14 three data-storage spaces 28 may be concatenated for receiving data signals. For a larger and newer DASD 14, five of the data-storage spaces 28 of cache 14 may be used, and so forth. In the latter instance the director 12 keeps a table (not shown) relating each DASD device address with a unit size of data transfer such that the appropriate number of data-storage spaces 28 may be selected for each receiving data-transfer operation. Some DASD 14s are operated in a front store/back store concept such that a portion of the DASD is addressable separately in a track subunit such as one-third or one-fourth of a track. Other tracks within the same DASD 14 may be addressed only as whole track units. In this case the same principles of the invention can be applied equally. Of course, segment counter 129 has to be adjusted accordingly.

Detailed Description Text (21):
To effect the deallocation of spaces 28, segment counter 129 supplies the number of segments over bus 145 that have been accessed in the current sequence of data transfer operations. Referring back momentarily to FIG. 3, step 110 determines the value of segment counter 129.

Detailed Description Text (22):
FIG. 5 illustrates the operation of directory 16. Directory 16 includes a plurality of registers, each of which is uniquely associated with one and only one of the data-storage spaces 28 of cache 15. Access to directory 16 is based upon a received DASD 14 address as received over bus 150 from using units 10 via programmed processor 17. A hash circuit 151 analyzes and parses the received DASD address into well known hash classes. The entire address base of all DASDs 14 of a particular data-storage system are divided into classes in accordance with track number, device number and DASD cylinder number (cylinders are all record tracks at one radial location or address). Each hash class has a single register in a scatter index table SIT 152. The output of hash circuit 151 addresses one and only one register in SIT 152. SIT 152 stores the address of a directory 16 register having a DASD address in its section 20 residing in the hash class defined for the given SIT 152 register. Such address is supplied, as indicated by arrow 154, for selecting the indicated one of the directory 16 registers. Within each directory 16 register is a hash pointner (HASH P) 155 which points to the next directory 16 register containing a DASD address within the same hash class. The last directory 16 register in the singly-linked list contains all zeros or a special code indicating it is at the end-of-chain. Accordingly, to scan the directory 16 registers, programmed processor 17 activates hashing circuit 151 for accessing the directory 16 registers from SIT 152 and accesses the first register for comparing the DASD address contained in its section 20 with the received DASD address on bus 150, as indicated by compare circuit 157. If there is a favorable compare, a cache hit has occurred as indicated by a signal on line 158. In the practical embodiment, line 158 is a logic path within programmed processor 17 in a program of instruction, such as program 36. A noncompare is indicated by numeral 159, then the hash pointer 155 is read and the directory 16 register pointed to by that hash pointer has its DASD address portion compared in a like manner. This cycle repeats until either a favorable compare indicates a cache hit or an end-of-chain (EOC) occurs. In the case of EOC, as indicated by numeral 160, a cache miss has occurred.

Detailed Description Text (23):
Each of the directory 16 addresses can contain a P1 pointer in section 22. As mentioned earlier, the actual address of the directory 16 register may be associated with a data-storage area 28 in a linear fashion. By having a P1 section 22 no ordered relationship between directory 16 structure and the organization of cache 15 is required.

Detailed Description Text (24):
LRU 47 also resides within directory 16. Each of the registers of directory 16 has a portion 167 which contains a pointer which points to a directory 16 register corresponding to a data-storage space 28 of cache 15 which is one less recently used than the data storage space pointed to by the current register. In a similar manner, section 168 has a more-recently-used pointer pointing to a directory 16 register corresponding to a data-storage space 28 which is more recently used. Accordingly, sections 167 and 168 are a doubly-linked list of directory 16 registers constituting an indication of the recentness of usage of the various data-storage areas. The least recently used data-storage areas represented by a special code in the LRUP area. While the most recently used data-storage areas indicated by a special code in the MRUP area.

Detailed Description Text (25):
Control store 18 of programmed processor 17 contains socalled LRU and MRU anchors 165 and 166. The LRU anchor 165 contains the address of a directory 16 register which is least recently used, while MRU anchor 166 points to the directory 16 register corresponding to the data-storage space 28 which is most recently used. The updating of the doubly-linked list 167, 168 and the anchors 165, 166 is well known and not described for that reason. When either P2 or P3 is set to zero, as in steps 91 or 93 of FIG. 3, director 17 updates the doubly-linked list 167, 168 by making the corresponding cache 15 segments free (F-bit 171 is set to unity) and relink the freed segments at the LRU end portion of the linked list. Additionally, when DASD 14 is not concurrently updated with data updates in cache 15, LRU 47 then includes a status indicator for the corresponding data-storage spaces 28. M-BIT 170 indicates whether or not the data contents of the corresponding data-storage space 28 has been modified by using unit 10. When M-BIT 170 is zero, the corresponding data-storage space is available for deallocation. Since no data transfer from cache 15 to DASD 14 is required for reallocation, when M-BIT 170 is equal to one (data in cache 15 has been changed), before the corresponding data-storage space is available for reallocation, the data contents of the corresponding data-storage space 28 has to be moved to the associated data-storage area of DASD 14. When DASD 14 and cache 15 are concurrently updated, M-BIT 170 is dispensed with. F-BIT 171 indicates whether or not the data-storage space 28 is free (unallocated) and available for allocation. The LRU scan described with respect to steps 101 and 95 of FIG. 3 begins with the LRU anchor 165 indicated directory 16 register and scans the registers for F-BIT 171 equal to 1 using the doubly-linked list 167 and 168. If the scan finds no free spaces from the F-BITs 171 indicated data-storage spaces, then a second scan for the M-BITs 170=0 is made. Of course, each directory 16 register contains additional control information as indicated by ellipsis 175.

Current US Original Classification (1):
707/205

CLAIMS:

1. The machine-implemented method of managing data-storage space in a data-storage unit for storing data receivable in relatively large address spaces and wherein the data contents in the address spaces are often much less than the data-storage capacity thereof;

including the machine-executable steps of:

establishing allocatable addressable data-storage segments within said data-storage unit that are a first submultiple of capacity of said storage space and each of said segments having a first address space less than each of said large address spaces;

receiving a request to store data in said data-storage unit which is receivable within a one of said large address spaces;

allocating a first number of said segments in response to said received request such that the total allocated first address space is not less than the data-storage capacity of said one large address space;

transferring data to said data-storage segments allocated for the data requested to be stored beginning with a first of said allocated segments, filling the first segment with a first portion of said data and storing remaining portion of said data in successive ones of said allocated segments until all data to be stored is in fact stored;

determining which, if any, of said allocated ones of said allocated segments are in fact storing data and which of said allocated segments are not storing data; and

deallocating those allocated segments which are not in fact storing data for later reallocation for storing other data yet to be received and outside said one large address space.

3. The machine-implemented method set forth in claims 1 or 2 wherein a source of data to be stored in said data-storage unit is from a direct access storage device having a plurality of addressable data-storage tracks, each of said tracks having a like data-storage capacity equal to the capacity of each of said large address spaces;

further including the steps of:

after allocating said allocated segments, supplying addresses of all of said allocated segments to a like plurality of address registers in said data-storage unit wherein each of said address registers stores the address of a respective one of said allocated segments; and

transferring data from said direct access storage device to said data-storage unit and switching addressing from a one of said address registers to another during said data transfer for not interrupting data transfer in switching from one of said allocated segments to another.

4. The machine-implemented method set forth in claim 1;

further including the steps of:

indicating that a transfer of data into said data-storage unit is less than the capacity of said first address space;

for each data transfer less than the capacity of said first address space, omitting said allocating, determining and deallocating steps set forth in claim 1.

5. The machine-implemented method set forth in claim 1 further including the machine-executable steps of:

indicating that a said large address space has a given extent of data therein and allocating a number of said segments in said allocating steps in accordance with said indicated extent such that a successive one of said data transfers may have a diversity of numbers of said segments allocated for such successive data transfer.

7. For use in a data-storage hierachy having a backing store coupled to a front store, each store having a plurality of addressable data-storage spaces, said hierarchy being adapted to be coupled to a host processor for transferring data signals therewith, means for coupling the host processor and said front and back stores together for transferring data signals therebetween;

the improvement including in combination;

means for indicating that data is to be written into said front store;

allocation means coupled to said indicating means for allocating a plurality of allocatable sets of said addressable data-storage spaces in said front store for said data indicated to be written therein, said allocation means also having means indicating that the allocatable unit of said backing store has a given multiple of data-storage spaces equal to the storage capacity of a given number of data-storage spaces in each set of data-storage spaces of said front store;

means coupled and being responsive to said allocation means for allocating said sets of addressable data-storage spaces and to activate said coupling means to transfer said indicated data into said front store for storage in said allocated sets;

examining means coupled to said front store for examining said allocated sets of data-storage spaces for determining which of said allocated data-storage spaces in fact received data signals during said indicated data transfer; and

deallocation means coupled to said examining means and to said front store for deallocating all of said data storage paces that in fact did not receive and store data during said indicated data transfer and that were allocated therefor.

9. The improvement set forth in claim 7 wherein said front store is a volatile random-access memory, said backing store is a plurality of addressable direct access storage devices wherein an addressable portion of said direct access storage devices is a given number of bytes of data which is a predetermined multiple of the data-storage capacity of each of said front store

addressable data-storage spaces;

directory means coupled to said front store and adapted for receiving address signals from a coupled host processor, said received address signals are for said direct access storage devices and means in said directory for directing access to said front store whenever data-storage spaces in said front store have been allocated to a corresponding received direct access storage device address; and

LRU means for indicating usage of said data-storage spaces and said allocation means and means coupled to said LRU means for selecting those data-storage paces of said front store having a least-recent usage for allocation to said indicated data transfer.

10. In a data-storage hierarchy having a front store and a backing store and adapted to be coupled to a host processor for transferring data signals therewith and having data-transfer means for transferring data signals between said host processor, said front store and said backing store;

a control data processor coupled to all of said stores and to said data transfer means and being adapted to receive commands from the host processor relating to movement of data in the data-storage hierarchy;

said backing store having a plurality of backing store addressable allocatable data-storage spaces each of which has a first data-storage capacity;

said front store having a plurality of front store addressable allocatable data-storage spaces each of which has a second data-storage capacity less than said first data-storage capacity;

a control store coupled to said control data processor for storing first program indicia for enabling said control data processor to indicate that data is to be stored in said front store in a data-storage operation;

second program indicia in said control store for enabling said control data processor to allocate a given number of said front store data-storage spaces for storage of data in said front store which in total yields an allocated data-storage capacity of not less than said first data-storage capacity;

third program indicia in said control store for enabling said control data processor to transfer and store data in said allocated front store data-storage spaces up to said first capacity;

fourth program indicia in said control store for enabling said control data processor upon completion of the data-storage operation to examine which of the allocated spaces allocated by said second program indicia have in fact received and stored or not received and not stored data as a result of the data-storage operation; and

fifth program indicia in said control store for enabling said control data processor to deallocate those second indicia allocated data-storage spaces of said front store not in fact storing data as a result of the data storage operation whereby the data-storage space of said front store may be more efficiently utilized.

Previous Doc Next Doc Go to Doc#

L5: Entry 3 of 8 File: USPT Jun 3, 2003

DOCUMENT-IDENTIFIER: US 6574699 B1
TITLE: Fast track reassign in a rotating storage media

Brief Summary Text (15):
The zone conversion unit 26 is used in disk drives implementing zone bit recording and typically uses a zone table including a zone entry for each zone. Each of the zone entries includes: a starting PBA for the corresponding zone; the number of data sectors per group, where a group is some fixed unit such as a track or cylinder; the number of data sectors per track; and the starting cylinder of the group. In such an implementation, a search for given PBA simply requires locating the appropriate zone entry in the table. Thus, if the group is defined as a cylinder, the zone table can easily be used to determine a PBA offset, a track offset, a head, a sector, and a cylinder. Alternatively, if the group is defined as a track, the table can easily be used to determine PBA offset, a track, a sector, a cylinder, and a head. Thus, regardless of the group definition, the PBA leads readily to a physical disk location. Once the disk drive completes the required seek operation to the cylinder and head identified, the drive formatter scans for either the desired LBA or the desired sector by examining the ID field of each data sector as it passes under the head, or calculates the actual location of the desired LBA (sector) using No-ID techniques. When the appropriate data sector is found, the data is transferred and the operation is complete.

Brief Summary Text (31):
Conventionally, a track detail object is generated for each track upon transfer of the data, or a portion of the data, stored in the track. ZCHS values are generated for the starting sector of the transfer. The length of the data transfer, which may span several tracks, is also received from the host. The track detail object is generated based on ZCHS values, defect information, and state information. Initially, the sector format conversion system 10 (FIG. 1) receives a logical block address from the host system, and the LBA to PBA conversion unit 12 (FIG. 1) determines a corresponding physical block address, sets pointers in the defects/spares tables, determines a next boundary LBA, and the distance to the next boundary LBA. In a detail state sub-process, the PBA to ZCHS conversion unit 12 determines corresponding ZCHS values based on the physical block address referring to the zone conversion unit 26 (FIG. 1). A map track sub-process is executed to build a track image based on the starting sector of the transfer, the number of sectors to be transferred, skip locations which will be dropped in the transfer, and state information required by a state machine (not shown) building the track details. A next track sub-process advances the physical state to the next head/track.

Brief Summary Text (32):
After the track detail object is generated, front end transfer hardware is armed, in accordance with a pipeline type operation, and the data is transferred. The front end transfer hardware includes a starting sector register for storing a starting sector value; a skew register for storing a skew value; a length-of-transfer register for storing a value indicating the total number of sectors to be transferred; and a first-in-first-out buffer (FIFO) for storing the skips information derived from the VT and VS tables of the defect mapping unit 12 (FIG. 1), which include the locations of defective sectors and spare sectors.

Brief Summary Text (40):
If the reassign sector sequentially precedes the corresponding spare sector, the step of reading data from the first trailing edge non-skip sector of the intermediate track includes: reading a track detail object associated with the intermediate track from a memory storage device of the disk drive system; post processing the track detail object so that it specifies a transfer of only the first trailing edge non-skip sector of the intermediate track; storing the track detail object in a track detail queue in a memory storage device of the disk drive system; setting a fast track feed mode enabling streaming of the track detail object from the

track detail queue; and reading the data into a data buffer of the disk drive system by transferring the data as specified by the track detail object.

Brief Summary Text (41):
If the spare sector sequentially precedes the corresponding reassign sector, the step of reading data from the first non-skip leading edge sector of the intermediate track includes: reading a track detail object associated with the intermediate track from a memory storage device of the disk drive system; post processing the track detail object so that it specifies a transfer of only the first leading edge non-skip sector of the intermediate track; storing the track detail object in a track detail queue in a memory storage device of the disk drive system; setting a fast track feed mode enabling streaming of the track detail object from the track detail queue; and reading the data into a data buffer of the disk drive system by transferring the data as specified by the track detail object. The foregoing and other objects, features, and advantages of the represent invention will be apparent from the following detailed description of the preferred embodiment which makes reference to the several figures of the drawing.

Detailed Description Text (51):
As mentioned above, the skew delta adjustment value associated with each intermediate track for the reverse reassign operation varies as a function of the positions of skip sectors in the forward direction relative to the trailing edge (logical sector n-1 position for a track having n sectors) of the tracks. For each of the intermediate tracks of a reverse reassign span, disposed between a reassign sector and a corresponding spare sector, the reassign operation provides a corresponding positive skew delta adjustment value for increasing the track skew factor associated with the track. By increasing a track skew factor by a corresponding skew delta adjustment value, the track is effectively rotated in a reverse direction by a number of sectors equal to the corresponding skew delta adjustment value. A corresponding skew delta adjustment values generated for each of the tracks is shown adjacent the rotated tracks, and the skewing of each of the tracks is depicted in linear increments each representing an increment of rotation.

Detailed Description Text (70):
The track detail object 420 further includes: a current state field 430; a starting sector field 432 for storing a starting sector value; a length of transfer field 434 for storing a value indicative of the number of blocks (or sectors) to transfer; a skip count field 436 for carrying a value indicative of the number of skip sectors on the corresponding track; and a field 438 for carrying a value indicative of a maximum number of skips sectors per track. Front end hardware (not shown) for transferring data stored on a track, or a portion of a track, includes: a starting sector register for receiving the starting logical sector value; a skew register for receiving a skew value; a length-of-transfer register for receiving the value indicative of the number of blocks to transfer; and a first-in-first-out buffer (FIFO) for receiving the skip locations which include the locations of defective sectors and spare sectors.

Detailed Description Text (71):
Conventionally, a track detail object is generated for each track upon transfer of the data stored in the track. A sector format conversion system receives a logical block address from the host system, and determines corresponding ZCHS values for the starting sector of the transfer. Relationship (2), below, yields a physical sector location based on a logical sector value, a total skew value, and a number of sectors per track.

Detailed Description Text (72):
The length of the data transfer, which may span several tracks, is also received from the host. Skip information for the transfer, derived from the VT and VS tables of the defect mapping unit 168 (FIG. 5), is loaded into the FIFO for storing the skips locations.

Detailed Description Text (73):
In one embodiment of the reassign process of the present invention, a track detail object 420 is generated for each track of a reassign span in accordance with a read traverse and filter sub-process wherein track details are accessed, post-processed where necessary, and then stored in a track detail queue for use in a subsequently executed fast streaming process; Data is not transferred until after the read traverse and filter sub-process. After the track detail objects for each track of the reassign span are queued in a buffer, the disk drive is set to

operate in a fast track feed mode wherein the track detail objects are streamed out of the buffer for a fast transfer which may be executed within the skew time associated with the corresponding track. In an alternative embodiment of the reassign process of the present invention, a high speed processor may be used to perform the necessary track detail calculations during a transfer "on the fly" wherein track detail objects need not be generated and stored for each track prior to the transfer.

Detailed Description Text (84):
It is then determined at 510 whether there are more tracks in the reassign span 450 (FIG. 13A), and if so, the process proceeds to step 512 to advance a pointer to a next track. From step 512, the process proceeds back to step 504 to fetch a track detail object for a current track, and then processes the track detail object in accordance with step 506 as described. Note that steps 504 through 512, which are repeated until the track detail queue includes a track detail object for each of the tracks of the forward reassign span 450 (FIG. 13A), are executed in a mode wherein data is not transferred after each track detail object is generated as in conventional transfer operations.

Detailed Description Text (91):
It is then determined at 510' whether there are more tracks in the reverse reassign span 470 (FIG. 13B), and if so, the process proceeds to step 512' to advance a pointer to a next track. From step 512', the process proceeds back to step 504' to fetch a track detail object for a current track, and then processes the track detail object in accordance with step 506' as described. Note that steps 504 through 512, which are repeated until the track detail queue includes a track detail object for each of the tracks of the reverse reassign span 470 (FIG. 13B), are executed in a mode wherein data is not transferred after each track detail object is generated as in conventional transfer operations.

Detailed Description Text (92):
After it is determined at 510' that there are no more tracks in the reverse reassign span, it is assumed that a track detail object for each track of the reassign span is queued in the buffer, and the process proceeds to step 514' in which the disk drive is set to operate in a fast track feed mode wherein the track detail objects previously stored in the buffer may be streamed out of the buffer for a fast transfer which may be executed within the skew time associated with the corresponding track. As mentioned above, in the alternative embodiment of the reassign process, a high speed processor may be used to perform the necessary track detail calculations during a transfer "on the fly" wherein track detail objects need not be generated and stored for each track prior to the transfer of data.

Detailed Description Text (103):
If it is determined at 624 that the current track is not an intermediate track, but is rather one of the first and second partial tracks of the reassign span, the process proceeds to step 626 in which virtual sector (VS) entries (associated with sectors greater than the spare sector and less than the reassign sector) of the defect map of the defect unit 168 (FIG. 5) are incremented for the current track. From step 626, the process proceeds to determine at 628 whether there are more tracks in the reverse reassign span 470 (FIG. 13B), and if so, the process proceeds to step 630 to advance a pointer to a next track of the reverse reassign span 470 (FIG. 13B). From step 630, the process proceeds back to step 622 to fetch a track detail object for a current track, and the track detail object is then processed in accordance with step 624 as described.

Detailed Description Text (116):
It is then determined at 708 whether there are more tracks in the reassign span, and if so, the process proceeds to step 710 to advance a pointer to a next track. From step 710, the process proceeds back to step 702 to fetch a track detail object for current track and process the track detail object in accordance with steps 704 and 707 as described. Note that steps 702 through 710, which are repeated until the track detail queue includes a track detail object for each of the tracks of the reassign operation, are executed in a mode wherein data is not transferred after each track detail object is generated as in conventional transfer operations.

Current US Cross Reference Classification (1):
707/205

CLAIMS:

6. A reassign process as recited in claim 4 wherein if said reassign sector sequentially precedes said corresponding spare sector, said step of reading data from said first trailing edge non-skip sector of said intermediate track comprises: reading a track detail object associated with said intermediate track from a memory storage device of said disk drive system; post processing said track detail object so that it specifies a transfer of only said first trailing edge non-skip sector of said intermediate track; storing said track detail object in a track detail queue in a memory storage device of said disk drive system; setting a fast track feed mode enabling streaming of said track detail object from said track detail queue; and reading said data into a data buffer of said disk drive system by transferring said data as specified by said track detail object.

7. A reassign process as recited in claim 4 wherein if said spare sector sequentially precedes said corresponding reassign sector, said step of reading data from said first non-skip leading edge sector of said intermediate track comprises: reading a track detail object associated with said intermediate track from a memory storage device of said disk drive system; post processing said track detail object so that it specifies a transfer of only said first leading edge non-skip sector of said intermediate track; storing said track detail object in a track detail queue in a memory storage device of said disk drive system; setting a fast track feed mode enabling streaming of said track detail object from said track detail queue; and reading said data into a data buffer of said disk drive system by transferring said data as specified by said track detail object.

13. A reassign process as recited in claim 11 wherein if said reassign sector sequentially precedes said corresponding spare sector, said step of reading data from said first trailing edge non-skip sector of said intermediate track comprises: reading a track detail object associated with said intermediate track from said memory storage means; post processing said track detail object so that it specifies a transfer of only said first trailing edge non-skip sector of said intermediate track; storing said track detail object in a track detail queue in said memory storage means of said disk drive system; setting a fast track feed mode enabling streaming of said track detail object from said track detail queue; and reading said data into said memory storage means by transferring said data as specified by said track detail object.

14. A reassign process as recited in claim 11 wherein if said spare sector sequentially precedes said corresponding reassign sector, said step of reading data from said first non-skip leading edge sector of said intermediate track comprises: reading a track detail object associated with said intermediate track from said memory storage means; post processing said track detail object so that it specifies a transfer of only said first leading edge non-skip sector of said intermediate track; storing said track detail object in a track detail queue in said memory storage means; setting a fast track feed mode enabling streaming of said track detail object from said track detail queue; and reading said data into said memory storage means by transferring said data as specified by said track detail object.

20. A machine readable storage device as recited in claim 18 wherein if said reassign sector sequentially precedes said corresponding spare sector, said step of reading data from said first trailing edge non-skip sector of said intermediate track comprises: reading a track detail object associated with said intermediate track from a memory storage device of said disk drive system; post processing said track detail object so that it specifies a transfer of only said first trailing edge non-skip sector of said intermediate track; storing said track detail object in a track detail queue in a memory storage device of said disk drive system; setting a fast track feed mode enabling streaming of said track detail object from said track detail queue; and reading said data into a data buffer of said disk drive system by transferring said data as specified by said track detail object.

21. A machine readable storage device as recited in claim 18 wherein if said spare sector sequentially precedes said corresponding reassign sector, said step of reading data from said first non-skip leading edge sector of said intermediate track comprises: reading a track detail object associated with said intermediate track from a memory storage device of said disk drive system; post processing said track detail object so that it specifies a transfer of only said first leading edge non-skip sector of said intermediate track; storing said track detail object in a track detail queue in a memory storage device of said disk drive system; setting a fast track feed mode enabling streaming of said track detail object from said track detail queue;

and reading said data into a data buffer of said disk drive system by transferring said data as specified by said track detail object.

Previous Doc Next Doc Go to Doc#

L7: Entry 3 of 5 File: USPT Feb 12, 1985

DOCUMENT-IDENTIFIER: US 4499539 A
TITLE: Method and apparatus for limiting allocated data-storage space in a data-storage unit

Abstract Text (1):
A data-storage buffer transfers data signals with other units in relatively large blocks of data. Such large blocks storable in large address spaces are not always filled with meaningful data. To more efficiently use the data-storage space in the data-storage buffer, the allocatable unit or segment of the data buffer is made smaller than the data capacity of the large block. Each time a large block of data is to be written into the data buffer, a sufficient number of the segments for storing data of one large block is allocated for receiving the data. After the data of the one block is written into the data buffer, the allocated segments are examined; all of the allocated segments not storing data from the one large block are deallocated. The invention is particularly useful for data buffers acting as cached data storage for large-capacity direct-access storage devices (DASD) and are coupled to host processors programmed to operate with such DASD. The procedure is followed for data written into the caching data storage whether supplied by DASD or the host processors.

Brief Summary Text (8):
The management of data-storage apparatus for ensuring full utilization of such space available in any data storage unit includes storing variable-length data. For example, U.S. Pat. No. 3,739,352, shows a microprogrammed processor associated with a so-called "free-field" memory in which operands of any length in terms of number of bits can be processed. The free-field memory is addressed by an address register that points to the boundary between any two bits stored in the memory as the start of a field and indicates the number of bits in the field up to a maximum bit capacity of the memory. While this technique certainly appears to provide for a maximal packing of a given memory (data-storage unit), when such data is replaced by other data the probability of the replacing data having an extent (number of bits) equal to the data being replaced is relatively small. This means that each time data is replaced that the memory must be reformatted if the storage efficiency is to be maintained. Accordingly, this technique, while probably valuable for many applications, is not applicable to a front-store/back-store data-storage hierarchy because of the data replacement operations. As a result of such a scheme, it can be easily envisioned that fragmentation of data would occur which requires extensive and time-consuming management techniques not desired in a peripheral data-storage hierarchy.

Brief Summary Text (12):
In a peripheral data-storage unit, U.S. Pat. No. 4,103,329 shows handling data represented by variable field length for using less data-storage. The bit fields are handled independently in the natural storage addressing elements and boundaries. This patent shows initializing a displacement register to contain an element displacement from a base address which contains the first bit of a desired bit field. While such a technique is certainly appropriate for packing data into a main memory for use by a host processor, the complexity and tracking of all of such data wherein the quantity of data is in the megabyte range becomes excessively expensive. Accordingly, these later-described techniques are also not fully satisfactory for managing a front store of a peripheral data-storage hierarchy.

Brief Summary Text (16):
In accordance with the invention, data-storage apparatus has allocatable data-storage spaces each having a data-storage capacity less than a maximal capacity required for given data transfers. Upon each incoming data transfer, an initial maximal allocation of data-storage space in the data-storage unit is made for the expected data. The data is then transferred to the allocated data-storage space. Upon completion of the data transfer, an examination is made of which allocated data-storage spaces actually received none of the incoming data. All of

those allocated data-storage spaces receiving no such data are then deallocated and allowed to be used for storing other data.

Brief Summary Text (17):
In another context, a maximal initial allocation of data storage space is made in accordance with the size of a maximal data transfer and upon completion of the data transfer the initial allocation is reduced to the size of the actual data transfer with all remaining portions of the allocated data-storage spaces being deallocated.

Detailed Description Text (3):
Directory 16 has a plurality of registers, as later detailed with respect to FIG. 5; each of which includes a stored DASD address (DASD ADDR) 20 which identifies the DASD 14 address which is intended to store or is actually storing data that is currently stored at an addressable portion of cache 15. The cache 15 storage location is indicated by cache address pointer P1 contained in section 22 of each directory 16 register or can be indicated by which directory 16 register is storing DASD ADDR; i.e. directory 16 register addresses are mapped to cache 15 addressable data-storage spaces. In accordance with the invention, cache 15 has allocatable data-storage spaces (sets of addressable data-storage register which are allocated as a single unit) having a capacity smaller than the data-storage capacity of a data-storage track 21 on DASD 14 pointed to by the DASD address 20 in directory 16. Preferably the capacity of the allocatable data-storage space in cache 15 is an integral submultiple of the maximum capacity of the data-storage track 21. For the present description the allocatable data-storage space of cache 15 is one-third the capacity of the DASD 14 track capacity. Three cache 15 addresses are required to address contents of a complete DASD track stored in cache 15. This addressing is achieved by having an address pointer P1 in section 22 of directory 16 identifying an allocatable data-storage space 23 of cache 15 for storing a first one-third portion of a DASD 14 track. In this one portion of allocatable data-storage space 23 are a pair of pointers P2, P3, respectively in spaces 24 and 26. P2 contains an address pointing to allocatable data-storage space 25 of cache 15 which stores the second one-third portion of track 14 while P3 points to an addressable, allocatable data-storage space 27 of cache 15 which stores the third one-third portion of track 14. In this manner the three allocatable data-storage spaces of cache 15 are concatenated to store the data contents of one track. It is to be understood that the number of said multiples of allocatable units in cache 15 may be a number other than three and that the additional pointers such as pointers P2 and P3 instead of being stored in the first allocatable space 23 pointed to by directory 16 may in fact be stored in area 22 along with pointer P1 in directory 16. Storing the addresses P2, P3 within cache 15 simplifies directory 16's structure; it does require one additional access to cache 15 for setting up data transfers, as will become apparent.

Detailed Description Text (4):
In accordance with the invention, when cache 15 is to receive data from either a using unit 10 or from DASD 14, programmed processor 17 will not have an indication of the extent of data which cache 15 will receive. That is, it may be a full track of data or less than a full track of data. Accordingly, each time cache 15 is to receive data, three allocatable data-storage spaces of cache 15 are allocated for the upcoming data transfer. Following such allocation, the data transfer ensues. Upon completion of the data transfer, programmed processor 17 examines which of the allocated data-storage spaces, such as 23, 25 and 27, have in fact received data for storage. Those allocated data-storage spaces of cache 15 not receiving any data during such data transfer are then deallocated, via appropriate pointers being zeroed and are made available for reallocation of data not related to the just-addressed DASD 14 track. In this manner, management of cache 15 data-storage space allows a greater number of tracks to be effectively stored in cache 15 with a relatively smaller cache capacity, i.e. reduces cost. In another view, if the same size cache 15 is used, then a greater performance is provided because the contents of a greater number of addressable DASD 14 data-storage tracks can be stored in cache 15. Cache 15, of course, has a large plurality of such allocatable data-storage spaces, as collectively indicated by numeral 28 and ellipsis 29.

Detailed Description Text (5):
The scattering of data from a single addressable data-storage track into a plurality of unrelated segments or data-storage spaces of cache 15 when high-speed data transfers are involved requires rapid concatenation and some buffering during the data transfer. For enhanced flow of data signals into and out of cache 15, as best seen in FIGS. 1 and 2, a plurality of system storageaddress registers 30 are provided. SSAR-0 receives pointer P1 from directory 16

in preparation for accessing allocated data-storage space 23. SSAR-1 and SSAR-2 respectively receive pointers P2 and P3 from areas 24, 26 of allocated data-storage space 23. This action completes the preparation for an ensuing data transfer. FIG. 4 shows how a plurality of address registers can quickly concatenate a plurality of addressable data-storage spaces for receiving a highspeed burst of data signals.

Detailed Description Text (6):
Upon each received data-access request, programmed processor 17 examines directory 16 to determine whether or not an associated allocatable data-storage space has been allocated to the DASD address received from using unit 10. If there is no match, then a cache-miss occurs, as indicated by numeral 35. Such a cache-miss can result in a data promotion from DASD 14 into cache 15, as will be later detailed. Such a miss activates processing unit 19 to access control store 18 for executing program 36 which may result in transferring data from DASD 14 to cache 15. Further, a cache write-hit, which indicates data will be transferred from a using unit 10 into cache 15, results in programmed processor 17 responding, as indicated by arrow 37, to use program 36 for preparing cache 15 to receive data from host 10, which may include up to a full track of data the extent of which, of course, is not presently known to the data-storage system. In such a host write, the host data is preferably simultaneously written to DASD 14.

Detailed Description Text (7):
In any event, for cache 15 to receive data without any overrun exposure results in programmed processor 17 in responding to program 36 to execute program 40 for allocating one DASD track capacity in cache 15 and setting the pointers P1, P2 and P3 as may be required. For example, if no cache 15 data-storage space has been allocated, then three cache 15 data-storage spaces are allocated with the corresponding pointers being generated. If on the other hand only one data-storage space is currently allocated, then two more data-storage spaces are allocated with the corresponding pointers being generated, all of which is detailed later with respect to FIG. 3. Upon completion of executing program 40, programmed processor 17 executes program 41 which actually causes the transfer of data to the cache 15 from either host 10 or DASD 14, as the case may be. Upon completion of the data transfer, programmed processor 17 checks the ending address of the last byte of data transferred into cache 15 by executing program 42. This check identifies which allocated data-storage spaces in fact received no data during the data transfer; i.e. the ending address check determines which of the three allocated data-storage space last received data. Then programmed processor 17 by executing program 43 deallocates any unused cache allocations made for the data transfer. Of course, preparatory to the execution of programs 36 and 43 and thereafter, other programs 44 which are commonly found in data-storage subsystems are executed. Since such programs do not have a bearing on an understanding of the present invention, they are not detailed.

Detailed Description Text (8):
In program 40 it may be required that programmed processor 17 replaced existing data in cache 15. Free or unallocated data-storage spaces must be identified. This action is achieved by an LRU (least recently used) replacement control list 47, usually found in data-storage hierarchies, explained with respect to FIG. 5. LRU 47 includes identification of those allocatable data-storage spaces which are available for allocation. Accordingly, execution of program 40 by programmed processor 17 results in usage of MRU-LRU program 46 for scanning LRU 47 to allocate data-storage spaces. If sufficient allocatable data-storage spaces are found, then those spaces are allocated with no further activity. However, if no allocatable data-storage spaces are found, then programmed processor 17 uses replace program 45 for transferring data from a replaced one of the allocated data-storage spaces to DASD 14 using known replacement techniques. When DASD 14 is updated concurrently with cache 15, the cache 15 space is immediately reallocated to the incoming data without any prereplacing data transfers to DASD 14. In this manner, cache 15 can be always filled with promoted data. Program 48 enables programmed processor 17 to access cache 15 using known techniques; accordingly, this program is not detailed.

Detailed Description Text (9):
FIG. 2 illustrates a preferred embodiment of the invention as employed in a two-storage director 12 data-storage arrangement. Each storage director 12 includes a plurality of so-called channel adaptors 50, also separately denominated as CAA through CAH, which connect the respective storage directors 12 to a plurality of using units 10 via a plurality of input/output connections 11. Each storage director 12 includes a programmed processor 17 which, as usual, includes a processing unit 19 having a control store 18 which contains computer

programs for performing the storagedirector functions. FIG. 2 shows the logical structure; i.e. the functions performed by processor 19 in executing the programs in control store 18. The programmed processor 17 includes programs constituting address and command evaluator ACE 52 which receive and evaluate using unit 10 supplied peripheral commands. Such functions are also performed in present day storage directors for noncached DASD as widely sold throughout the world and are a part of other programs 44 in FIG. 1. The programmed processor 17 also includes programs for direct access control DAC 53 which responds to commands evaluated and decoded by ACE 52 to control data transfers between using units 10 and DASD 14, as well as providing device commands to DASD 14 for performing well known DASD access and control functions. DAC 53 includes program 41 as well as programs for accessing DASD 14 included in other programs 44 relating to accessing DASDs 14 and transferring data between using units 10 and DASDs 14, all of which is well known. Programmed processor 17 further includes programs CAC 54 which is a cache access control for accessing cache 15. CD latches 59, one for each of the DASDs 14, are accessed by DAC 53 and CAC 54 respectively for determining whether to access cache 15 or DASD 14 directly and for setting the latches to D upon a cache miss. Connections from storage director 12 to DASDs 14 are via DASD circuits 55 which are constructed using known device adaptor and data-flow design techniques. Cache 15 is accessed via memory circuits (MEM CCTS) 56 which includes those circuits for generating addresses and access requests including SSARs 30. Cache 15 is a portion of a large random-access store 57, hereinafter referred to as a system store. It is preferred that cache 15 can simultaneously and independently handle data transfers with a DASD 14 and a host 10. The directory 16 and LRU 47 for cache 15 are also stored in system store 57. Additionally, any using unit 10 can command the storage directors 12 to keep data in cache, i.e. pin or bind the data to cache 15. For all bound tracks, it records a cache bound list 60, stored within directory 58 but shown separately for clarity, indicates to both storage directors 12 which data stored in cache 15 is to remain in cache 15. Such bound data is not listed in LRU 47 for preventing replace program 45 from reallocating cache 15 space.

Detailed Description Text (11):
Programmed processor 17 at 70 receives a storage-access request. This request is decoded and evaluated in ACE 52 using known techniques. At 71, programmed processor 17 DAC 53 portion examines the CD latch 59 (FIG. 2) related to the DASD 14 addressed in the received storage-access request to determine whether cache 15 or only DASD 14 to the exclusion of cache 15 is to be accessed. For a direct access, DASD 14 is accessed at 72 using usual DASD access methods. For a cache C access, programmed processor 17 searches directory 16 at 73 to determine whether or not the track requested in the received storage-access request (I/O command) has allocated space in cache 15. In this regard it is noted that some commands will require a direct connection to DASD 14 to the exclusion of cache 15. Accordingly, ACE 52 in detecting such a received I/O command sets latch 59 for the addressed DASD 14 to the direct mode "D". An example of such an I/O command is to recalibrate a DASD 14. Searches and SEARCH ID EQUAL commands can be performed for cache 15 accesses within directory 16, i.e. the commands are performed in a virtual manner not involving DASD 14. In the preferred embodiment directory 16 does not separately identify records in a track; only tracks are identified, no limitation thereto intended. Upon completion of the directory 16 search, programmed processor 17 at 74 determines whether or not a cache-hit has occurred. If a cache-hit occurred, which is preferred programmed processor 17 at 74A transfers the P1 stored in section 22 of the directory 16 register identified by the received DASD 14 address to SSAR-0; then it transfers P2 and P3 respectively to SSAR-1 and SSAR-2 from their respective storage locations.

Detailed Description Text (12):
At step 75, director 12 examines cache 15 to determine whether or not the record to be accessed is stored in cache 15 (record hit). If the addressed record is in cache 15 (record hit is yes), then additional segments may not be needed to successfully complete the ensuing data transfer. Then at step 76 the type of data transfer operation to be performed is examined. For a read operation R (transfer of data to a host 10), director 12 at step 77 transfers the requested data from cache 15 to the requesting host 10. Such transfer completes the operation permitting director 12 to exit the machine operation at 78 for performing other data processing operations. For a write operation W (transfer of data from a host 10) indicated at step 76, director 12 at step 80 examines the received host 10 supplied command for ascertaining if the write is a FORMAT write (access to DASD 14 is requested to the exclusion of cache 15) or any other form of write (cache 15 is to be utilized) is requested. For a FORMAT write, director 12 in step 81 deallocates any allocated cache 15 data-storage space and transfers the received data to DASD 14. For a nonformat write (FORMAT=0) at step 80, director 12 in step 82 transfers data from the requesting host 10 to both cache 15 and DASD 14 respective addressed data-storage

areas. In this manner, cache 15 and DASD 14 always have identical copies of the same data. From steps 81 and 82, director 12 proceeds to other data processing operations via logic path 78. This operation allows less than a full track allocation in cache to handle successive data transfers (partial track allocations).

Detailed Description Text (13):
Returning to step 75, when director 12 does not find the addressed record (record hit is no), then for the impending data transfer to the cache, additional segments may be allocated for the ensuing data transfer. In steps 85 and 86 director 12 examines the values of P2 and P3. For either or both pointers being zero (no corresponding space has been allocated in cache 15), director 12 in step 86 allocates an additional segment to the track, as previously described and then proceeds to transfer data to cache 15 at 87. The step 87 data transfer can be a write from a host 10 to DASD 14 and cache 15, a read from DASD 14 to cache 15 and a host 10, or a staging data operation from DASD 14 to cache 15.

Detailed Description Text (14):
The post-transfer machine operations find director 12 examining cache 15 to determine which of the three allocated segments in fact received data from the just-completed data transfer. In step 88, director 12 examines a later-described "k-counter" 129 (FIG. 4) to ascertain the values 1, 2 or 3 which respectively indicate that one, two or three allocated segments (corresponding to P1, P2, P3) in fact received and are currently storing data. For k=1, director 12 in step 90 takes the segments 2 and 3 (also termed XM and YM, respectively) identifications and inserts same into the LRU list for making these segments available for allocation. In step 91, the corresponding pointers P2 and P3 are set to zero. For a value k=2 in step 88, director 12 in steps 92 and 93 inserts the third segment YM into the LRU list and sets pointer P3 to zero. For a value of k=3 in step 88, director 12 knows that all three segments have received and are currently storing data, hence it proceeds directly to do other data processing operations through logic path 78. Path 78 is also entered from steps 91 and 93, as well.

Detailed Description Text (15):
For a cache miss at 74 (hit=0), director 12 in steps 95 and 96 allocates three segments in cache 15 (XY, XM, YM) for the ensuing data transfer to cache 15 and sets the corresponding pointers P1, P2 and P3 in the respective SSAR's 0, 1 and 2. Then director 12 proceeds to the data transfer operation performed in step 87, as previously described.

Detailed Description Text (16):
In one embodiment of directory 16, each of the registers in directory 16 corresponded to a space 28 in cache 15. Hence, area 22 is dispensed with the register address within directory 16 also indicating (using base plus offset addressing) the beginning address in cache 15 of an associated space. Allocation then consists of inserting the appropriate DASD address in section 20 of such register. Addresses XM and YM respectively become pointers P2 and P3 and are stored in areas 24, 26 of the area 23 corresponding to address XY. Note there are no changes in directory 16 for these last two pointers. In the event that the last above-described directory 16 structures wherein a given register always is associated with a data-storage area of cache 15, then addresses P2 and P3 are inserted in these respective directory 16 registers; the registers for P2 and P3 are then omitted from LRU 47. The above completes setting up the pointers for the ensuing data transfer.

Detailed Description Text (17):
FIG. 4 illustrates cache 15 addressing circuits usable with the present invention. The data paths 100 extend from cache 15 through memory circuits 56 thence to DASD circuits 55 for data transfers with DASD 14. The data paths also extend to the channel adaptors 50 for data transfers with using unit 10, all as shown in FIG. 2. The data transfers between cache 15, adaptors 50 and DASD circuits 55 are under the control of usual automatic data-transfer circuits of known design and of current use in DASD storage systems. Such automatic transfer control circuits are shown as autocontrol 101 in FIG. 4 as being a part of memory circuits 56. Programmed processor 17 supplies a suitable start signal over line 102 to autocontrol 101. The description assumes the storage system address registers 30 have been loaded with the appropriate addresses P1, P2 and P3 received from programmed processor 17 respectively over address busses 110, 111 and 112. Such loading of address registers by a programmed processor is well known. Once autocontrol 101 receives the START signal, it supplies a cache 15 access enabling signal over line 103 to cache 15. As a result, cache 15 will receive addresses as

later described for accessing data-storage registers within the cache. The access-control signal on line 103 will carry an indication of whether the operation is a read-from-cache operation or a write-to-cache operation. Many caches 15 contain known refresh circuits which interleave refresh cycles with data-access cycles. Each time cache 15 transfers a set of data signals over data path 100, it indicates a cycle of operation to autocontrol 101 over line 104. Autocontrol 101 has been preset in a known manner for transferring a given number of data signals between cache 15 and either DASD 14 or host 10. When data signals are being written into cache 15, autocontrol 101 may not know the number of signals to be received. In this instance, a second signal is supplied over start line 102 to turn autocontrol 101 off for removing the signal on line 103. For example, in a host-to-cache data transfer, the host using the IBM 370 interface architecture can send a socalled COMMAND OUT I/O tag signal indicating the end of the data transfer. Such I/O tag signal results in programmed processor 17 sending a second signal on start line 102 to indicate to autocontrol 101 to terminate the data transfer. Termination of the data transfer either internally to autocontrol 101 or to externally received commands is indicated to programmed processor 15 by an END signal supplied over line 105. For each cycle of cache 15 operation, autocontrol 101 emits an address incrementing signal over line 115. The incrementing signal goes to one and only one of the SSARs 30 as selected by an SSAR address received over bus 130 from programmed processor 17. The addressing of a plurality of address registers is well known and not described for that reason. When P1, P2 and P3 are loaded, the SSAR address signals received from programmed processor 17 will select SSAR-0. Decoder 131 decodes the address signal and supplies an AND circuit enabling signal over line 132 to AND circuits 116 and 120; AND circuit 116 passes the address incrementing signal on line 115 to SSAR-0 for incrementing the address contained therein. Decrementing can be used as well. Each time SSAR-0 is incremented, it supplies a set of address signals to AND circuits 120 for transmitting same over the address bus 123 to cache 15 for selecting the next data-storage location within the cache 15 addressed data-storage space for the data transfer. When the SSAR-0 has counted through all of the addressable data-storage locations within one data-storage space 28 of cache 15, it supplies a carry signal over line 126 through OR circuit 125 for incrementing segment counter 129. Segment counter 129, which counts segments having 2.sup.k data-storage locations (k is an integer), has been preset to zero through a reset signal received from programmed processor 17 overline 140. Segment counter 129 supplies a zero signal over line 141 to decoder 131 for passing the received SSAR-0 address signal to decoding circuits resulting in the line 132 and enabling signal. When segment counter 129 is incremented by the SSAR-0 carry signal, it then supplies a one signal over line 142 to decoder 131.

Detailed Description Text (18):
Decoder 131 is of the type that can add one to the received SSAR address such that the line 132 AND-circuit enabling signal is removed and a new AND-circuit enabling signal is supplied over line 133. Such signal enables AND circuits 117 and 121 associated with SSAR-1. AND circuit 117 enables the address-incrementing signal on line 115 to increment SSAR-1 and then to supply address signals through AND circuits 121 to cache 15. In a similar manner SSAR-1 supplies its carry signal over line 127 to also increment segment counter 129 resulting in a two signal being supplied over line 143 to decoder 131. This causes decoder 131 to add two to the received SSAR address resulting in an AND-circuit enabling signal being sent only over line 134 to AND circuits 118 and 122 associated with SSAR-2. SSAR-2 then receives the address-incrementing signal and supplies the cache data-storage location signals to cache 15 for the third data-storage space being addressed in the sequence of operation.

Detailed Description Text (19):
Segment counter 129 is not restricted to counting segments of 2.sup.k sizes. By providing a segment size register, the counter 129 can count segments having any arbitrary size or variably sized segments. For simplicity segment sizes of 2.sup.k are preferred.

Detailed Description Text (20):
It is to be appreciated that a larger plurality of SSARs 30 may be provided, as indicated by ellipsis 147. As such, any three of the larger plurality of storage address registers may be used in sequencing cache 15 operation in accordance with the invention. Accordingly, there are a like greater plurality of AND circuits enabling lines indicated by ellipsis 148. In any event, the first storage address register which receives P1 is selected by programmed processor 17 in the usual manner. Programmed processor 17 then indicates which SSAR received the P1 address which starts a sequence of concatenated addresses within cache 15 for successively-accessed data-storage spaces 28. Accordingly, a variable number of data-storage spaces 28 can be used with a diversity of sizes of address spaces for receiving data signals. For example, if

two types of DASD 14s are attached to the directors 12, two different sizes of data transfer units (data contents of two DASD tracks have different numbers of stored data bits) may be involved. From a first DASD 14 three data-storage spaces 28 may be concatenated for receiving data signals. For a larger and newer DASD 14, five of the data-storage spaces 28 of cache 14 may be used, and so forth. In the latter instance the director 12 keeps a table (not shown) relating each DASD device address with a unit size of data transfer such that the appropriate number of data-storage spaces 28 may be selected for each receiving data-transfer operation. Some DASD 14s are operated in a front store/back store concept such that a portion of the DASD is addressable separately in a track subunit such as one-third or one-fourth of a track. Other tracks within the same DASD 14 may be addressed only as whole track units. In this case the same principles of the invention can be applied equally. Of course, segment counter 129 has to be adjusted accordingly.

Detailed Description Text (21):
To effect the deallocation of spaces 28, segment counter 129 supplies the number of segments over bus 145 that have been accessed in the current sequence of data transfer operations. Referring back momentarily to FIG. 3, step 110 determines the value of segment counter 129.

Detailed Description Text (22):
FIG. 5 illustrates the operation of directory 16. Directory 16 includes a plurality of registers, each of which is uniquely associated with one and only one of the data-storage spaces 28 of cache 15. Access to directory 16 is based upon a received DASD 14 address as received over bus 150 from using units 10 via programmed processor 17. A hash circuit 151 analyzes and parses the received DASD address into well known hash classes. The entire address base of all DASDs 14 of a particular data-storage system are divided into classes in accordance with track number, device number and DASD cylinder number (cylinders are all record tracks at one radial location or address). Each hash class has a single register in a scatter index table SIT 152. The output of hash circuit 151 addresses one and only one register in SIT 152. SIT 152 stores the address of a directory 16 register having a DASD address in its section 20 residing in the hash class defined for the given SIT 152 register. Such address is supplied, as indicated by arrow 154, for selecting the indicated one of the directory 16 registers. Within each directory 16 register is a hash pointner (HASH P) 155 which points to the next directory 16 register containing a DASD address within the same hash class. The last directory 16 register in the singly-linked list contains all zeros or a special code indicating it is at the end-of-chain. Accordingly, to scan the directory 16 registers, programmed processor 17 activates hashing circuit 151 for accessing the directory 16 registers from SIT 152 and accesses the first register for comparing the DASD address contained in its section 20 with the received DASD address on bus 150, as indicated by compare circuit 157. If there is a favorable compare, a cache hit has occurred as indicated by a signal on line 158. In the practical embodiment, line 158 is a logic path within programmed processor 17 in a program of instruction, such as program 36. A noncompare is indicated by numeral 159, then the hash pointer 155 is read and the directory 16 register pointed to by that hash pointer has its DASD address portion compared in a like manner. This cycle repeats until either a favorable compare indicates a cache hit or an end-of-chain (EOC) occurs. In the case of EOC, as indicated by numeral 160, a cache miss has occurred.

Detailed Description Text (23):
Each of the directory 16 addresses can contain a P1 pointer in section 22. As mentioned earlier, the actual address of the directory 16 register may be associated with a data-storage area 28 in a linear fashion. By having a P1 section 22 no ordered relationship between directory 16 structure and the organization of cache 15 is required.

Detailed Description Text (24):
LRU 47 also resides within directory 16. Each of the registers of directory 16 has a portion 167 which contains a pointer which points to a directory 16 register corresponding to a data-storage space 28 of cache 15 which is one less recently used than the data storage space pointed to by the current register. In a similar manner, section 168 has a more-recently-used pointer pointing to a directory 16 register corresponding to a data-storage space 28 which is more recently used. Accordingly, sections 167 and 168 are a doubly-linked list of directory 16 registers constituting an indication of the recentness of usage of the various data-storage areas. The least recently used data-storage areas represented by a special code in the LRUP area. While the most recently used data-storage areas indicated by a special code in the MRUP area.

Detailed Description Text (25):
Control store 18 of programmed processor 17 contains socalled LRU and MRU anchors 165 and 166. The LRU anchor 165 contains the address of a directory 16 register which is least recently used, while MRU anchor 166 points to the directory 16 register corresponding to the data-storage space 28 which is most recently used. The updating of the doubly-linked list 167, 168 and the anchors 165, 166 is well known and not described for that reason. When either P2 or P3 is set to zero, as in steps 91 or 93 of FIG. 3, director 17 updates the doubly-linked list 167, 168 by making the corresponding cache 15 segments free (F-bit 171 is set to unity) and relink the freed segments at the LRU end portion of the linked list. Additionally, when DASD 14 is not concurrently updated with data updates in cache 15, LRU 47 then includes a status indicator for the corresponding data-storage spaces 28. M-BIT 170 indicates whether or not the data contents of the corresponding data-storage space 28 has been modified by using unit 10. When M-BIT 170 is zero, the corresponding data-storage space is available for deallocation. Since no data transfer from cache 15 to DASD 14 is required for reallocation, when M-BIT 170 is equal to one (data in cache 15 has been changed), before the corresponding data-storage space is available for reallocation, the data contents of the corresponding data-storage space 28 has to be moved to the associated data-storage area of DASD 14. When DASD 14 and cache 15 are concurrently updated, M-BIT 170 is dispensed with. F-BIT 171 indicates whether or not the data-storage space 28 is free (unallocated) and available for allocation. The LRU scan described with respect to steps 101 and 95 of FIG. 3 begins with the LRU anchor 165 indicated directory 16 register and scans the registers for F-BIT 171 equal to 1 using the doubly-linked list 167 and 168. If the scan finds no free spaces from the F-BITs 171 indicated data-storage spaces, then a second scan for the M-BITs 170=0 is made. Of course, each directory 16 register contains additional control information as indicated by ellipsis 175.

Current US Original Classification (1):
707/205

CLAIMS:

1. The machine-implemented method of managing data-storage space in a data-storage unit for storing data receivable in relatively large address spaces and wherein the data contents in the address spaces are often much less than the data-storage capacity thereof;

including the machine-executable steps of:

establishing allocatable addressable data-storage segments within said data-storage unit that are a first submultiple of capacity of said storage space and each of said segments having a first address space less than each of said large address spaces;

receiving a request to store data in said data-storage unit which is receivable within a one of said large address spaces;

allocating a first number of said segments in response to said received request such that the total allocated first address space is not less than the data-storage capacity of said one large address space;

transferring data to said data-storage segments allocated for the data requested to be stored beginning with a first of said allocated segments, filling the first segment with a first portion of said data and storing remaining portion of said data in successive ones of said allocated segments until all data to be stored is in fact stored;

determining which, if any, of said allocated ones of said allocated segments are in fact storing data and which of said allocated segments are not storing data; and

deallocating those allocated segments which are not in fact storing data for later reallocation for storing other data yet to be received and outside said one large address space.

3. The machine-implemented method set forth in claims 1 or 2 wherein a source of data to be stored in said data-storage unit is from a direct access storage device having a plurality of addressable data-storage tracks, each of said tracks having a like data-storage capacity equal to the capacity of each of said large address spaces;

further including the steps of:

after allocating said allocated segments, supplying addresses of all of said allocated segments to a like plurality of address registers in said data-storage unit wherein each of said address registers stores the address of a respective one of said allocated segments; and

transferring data from said direct access storage device to said data-storage unit and switching addressing from a one of said address registers to another during said data transfer for not interrupting data transfer in switching from one of said allocated segments to another.

4. The machine-implemented method set forth in claim 1;

further including the steps of:

indicating that a transfer of data into said data-storage unit is less than the capacity of said first address space;

for each data transfer less than the capacity of said first address space, omitting said allocating, determining and deallocating steps set forth in claim 1.

5. The machine-implemented method set forth in claim 1 further including the machine-executable steps of:

indicating that a said large address space has a given extent of data therein and allocating a number of said segments in said allocating steps in accordance with said indicated extent such that a successive one of said data transfers may have a diversity of numbers of said segments allocated for such successive data transfer.

7. For use in a data-storage hierachy having a backing store coupled to a front store, each store having a plurality of addressable data-storage spaces, said hierarchy being adapted to be coupled to a host processor for transferring data signals therewith, means for coupling the host processor and said front and back stores together for transferring data signals therebetween;

the improvement including in combination;

means for indicating that data is to be written into said front store;

allocation means coupled to said indicating means for allocating a plurality of allocatable sets of said addressable data-storage spaces in said front store for said data indicated to be written therein, said allocation means also having means indicating that the allocatable unit of said backing store has a given multiple of data-storage spaces equal to the storage capacity of a given number of data-storage spaces in each set of data-storage spaces of said front store;

means coupled and being responsive to said allocation means for allocating said sets of addressable data-storage spaces and to activate said coupling means to transfer said indicated data into said front store for storage in said allocated sets;

examining means coupled to said front store for examining said allocated sets of data-storage spaces for determining which of said allocated data-storage spaces in fact received data signals during said indicated data transfer; and

deallocation means coupled to said examining means and to said front store for deallocating all of said data storage paces that in fact did not receive and store data during said indicated data transfer and that were allocated therefor.

9. The improvement set forth in claim 7 wherein said front store is a volatile random-access memory, said backing store is a plurality of addressable direct access storage devices wherein an addressable portion of said direct access storage devices is a given number of bytes of data which is a predetermined multiple of the data-storage capacity of each of said front store

addressable data-storage spaces;

directory means coupled to said front store and adapted for receiving address signals from a coupled host processor, said received address signals are for said direct access storage devices and means in said directory for directing access to said front store whenever data-storage spaces in said front store have been allocated to a corresponding received direct access storage device address; and

LRU means for indicating usage of said data-storage spaces and said allocation means and means coupled to said LRU means for selecting those data-storage paces of said front store having a least-recent usage for allocation to said indicated data transfer.

10. In a data-storage hierarchy having a front store and a backing store and adapted to be coupled to a host processor for transferring data signals therewith and having data-transfer means for transferring data signals between said host processor, said front store and said backing store;

a control data processor coupled to all of said stores and to said data transfer means and being adapted to receive commands from the host processor relating to movement of data in the data-storage hierarchy;

said backing store having a plurality of backing store addressable allocatable data-storage spaces each of which has a first data-storage capacity;

said front store having a plurality of front store addressable allocatable data-storage spaces each of which has a second data-storage capacity less than said first data-storage capacity;

a control store coupled to said control data processor for storing first program indicia for enabling said control data processor to indicate that data is to be stored in said front store in a data-storage operation;

second program indicia in said control store for enabling said control data processor to allocate a given number of said front store data-storage spaces for storage of data in said front store which in total yields an allocated data-storage capacity of not less than said first data-storage capacity;

third program indicia in said control store for enabling said control data processor to transfer and store data in said allocated front store data-storage spaces up to said first capacity;

fourth program indicia in said control store for enabling said control data processor upon completion of the data-storage operation to examine which of the allocated spaces allocated by said second program indicia have in fact received and stored or not received and not stored data as a result of the data-storage operation; and

fifth program indicia in said control store for enabling said control data processor to deallocate those second indicia allocated data-storage spaces of said front store not in fact storing data as a result of the data storage operation whereby the data-storage space of said front store may be more efficiently utilized.

Previous Doc Next Doc Go to Doc#